UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

New Simulation Techniques for Energy Aware Cloud Computing Environments

Autor:
Gabriel González Castañé

Directores:
Jesús Carretero Pérez
Alberto Núñez Covarrubias

DEPARTAMENTO DE INFORMÁTICA

Leganés, Abril de 2015

TESIS DOCTORAL

New Simulation Techniques for Energy Aware Cloud Computing Environments

Autor:
Gabriel González Castañé
Directores:
Jesús Carretero Pérez
Alberto Núñez Covarrubias

Firma del tribunal calificador:

Presidente:	Firma:
Vocal:	Firma:
Secretario:	Firma:

Calificación:

Leganés, a __ de _____ de _____.

”Adversity has the effect of eliciting talents which,
in prosperous circumstances, would have lain dormant.”

– Quintus Horatius Flaccus, Roman poet.

”La adversidad tiene el don de despertar los talentos,
que en la comodidad hubieran permanecido dormidos.”

– Quintus Horatius Flaccus, poeta Romano.

”Begin at the beginning,”
the King said, very gravely,
”and go on till you come to the end: then stop.”

– Lewis Carroll, Alice in Wonderland

”Empieza por el principio,”
contestó gravemente el rey,
”y sigue hasta que llegues al final: Entonces te detienes.”

– Lewis Carroll, Alicia en el país de las maravillas.

Agradecimientos

Esta ”sección” es la última que escribo del libro. En ella me gustaría tener unas palabras para todos aquellos que han sido capaces de aguantarme durante el transcurso de esta tesis.

Principalmente quiero dedicarle esta tesis a Patricia, mi princesa. Gracias. No puedo expresar escribiendo todo lo que has hecho por mí (y soportado) estos años. Una gran parte de esta tesis es tuya, porque sin tu apoyo esto no hubiera sido posible. Todos esos días y noches trabajando y tú comprendiéndolo siempre con una sonrisa en la cara, incluso en mis días de peor “genio” (que no han sido pocos). No tienes un lugar en mi corazón, lo tienes todo.

Mis padres, por supuesto. Qué sería de mi sin ellos. Gracias por vuestro apoyo. Ya la he acabado!!. Ahora qué me vaís a preguntar cuando me veais, sin poderme decir: hijo, ¿cuándo acabas la tesis?. Os quiero mucho, y no os cambiaba por nada del mundo.

Mi hermano, Eva, y Lucía. Gracias por escucharme cuando me he quejado, y por ser comprensivos con mi cabeza a 1000 cosas, con mis olvidos y sin tener tiempo para nada ni nadie. Con respecto a Lucía, imagino tu cara cuando seas capaz de ojear este libro y digas: ”dios mío, si un teléfono movil a día de hoy hace lo mismo”. En fin, es lo que tiene la tecnología, todo evoluciona muy rápido ;-).

Feli y Jose Luis, los padres de Patri, también tienen su hueco aquí. Por las veces que he estado con vosotros y me habéis escuchado. Por echarnos un cable en todo lo que habéis podido y más. Al final parece que todo llega. A ver cuándo llega ese Mercedes, y prendo fuego al Clio!.

A mis directores de tésis, Jesús Carretero y Alberto Núñez, por vuestro apoyo y consejo para hacer esta tesis. Os agradezco que hayáis confiado en mi para realizar la tesis.

A Jose Alberto (alias Rumano) y Javier (alias Jal). Siento haberos descuidado en el último período de la tesis, pero no he tenido tiempo para nada. Hay que retomar las viejas costumbres que ya van apareciendo nuevas generaciones de consolas. Echo de menos esos momentos de comidas de lujo ”C”, donde te ”invitabas” Rumano. A ver cuándo lo repetimos. Jal, poco más que decirte que ”fuá!”.

A mi amigo de toda la vida Jesus Plaza. No pensarías que me iba a olvidar de tí verdad?, Gracias por todos esos intentos de sacar dinero hasta de debajo de las piedras para que me fuera posible acabar la tesis y trabajar juntos. Eres un crack, y al final te veo en una mansión con un Ferrari!, eso si, tenemos una comida/cena pendiente que no se me olvidará.

A la gente de la complu, Manuel y Mercedes. Gracias por las reuniones, y los consejos, de corazón. Por las veces que hemos comido juntos y por el viaje a San Sebastián. La verdad es que ha sido de los mejores viajes de mi vida, justo cuando más lo necesitaba en la tesis.

Fueron 3 días que no deje de reírme ni un solo instante. Esas discusiones científicas entre metaleros vascos un martes a las 2 de la madrugada, no tiene precio. Y Meeeeeercedes, no te enfades conmigo, que ya acabo!.

A mis amigos de la Politécnica.

Pilar, e Isabel, por vuestro apoyo. Seguir siempre siempre igual de simpáticas.

A Norberto, porque tenemos muchas cosas pendientes, proyectos, robots (:D), y por tu apoyo. Ahora ya no podemos decir eso de: 'Ánimo doctor!'. Habrá que buscar algún proyecto de esos de construcción de Wall-e pero que dispare rayos laser o algo así.

En especial a tí Jorge. Que puedo decirte aquí. Para mi eres uno de mis mejores amigos, la verdad. Gracias por todos los consejos y tu ayuda. En especial porque cuando más lo necesitaba me ayudaste, cuando muchos otros no. Para mi significó mucho. Por cierto, tengo presente que te debo un cordero asado, varias botellas de vino, y una crema catalana. No dudes ni un instante que te lo pagaré, porque no lo pienso a olvidar.

A la gente de Granada. Use, Antoñico, Pereira, Ali, y Pablo. Vaya equipo montamos en un rato. Seguir así, sois unos cielos. Gracias por sacarme a tomar esas tapas y por enseñarme los rincones más lindos de esa ciudad. A cada cuál más especial. La verdad es que se echa mucho de menos. Pedir unos albondigones a mi salud!!

A Chema, por todos esos desayunos y conversaciones trascendentales que hemos tenido en Granada, que me servían para relajarme.

A Álvaro (alias el Mago, Malka ..). Por todos esos grandes fines de semana en el pasillo de los bajos, riéndonos de todo el que pasaba por delante. Iré a verte en cuanto busque hueco, que no me pierdo un espectáculo tuyo ni de broma. Seguramente hayas mejorado bastante, pero seguiré intentando desvelar los trucos en mi cabeza. Llegarás a ser un grande en la magia, estoy seguro.

A la gente de la Carlos III, de ARCOS.

Pablo y Gonzalo, mis ex-compañeros de despacho. Se os echa de menos con esas risas y locuras, aventuras de Pablo con los zombies, e historias varias de foros, gifs, bolitas de patata en el buffe de guarniciones (llegué a tener pesadillas con ellas) ... Hay que hacerse un Foster's en cuanto se pueda.

A Alberto García y a Carlos Gomez. Gracias por echarme un cable en la recta final Carlos. Te debo unas ¿Bulmer? en cuanto vengas de visita a mi casa. No te preocupes, que no te haré dormir conmigo (aunque se que lo deseas). Garci, debajo de esa capa de soldado del Red Army, eres un cielo. Cuanto me acuerdo de esos sahorma crispi, de los bailes regionales en corro, gogoasas, etc, no puedo dejar de reírme. Sobre todo con la historia del pinchazo de camino a visitar a Vlad. Tienes una historia que contar a tus hijos! Qué bien lo pasé en la estancia. Muchas gracias. En cuanto acabe hay que celebrarlo ;-).

Obviamente, Alejandro Calderón. Alex, infinito + 10 gracias por todo. Eres una gran persona, de veras. Por tus consejos, conversaciones, ayuda, gracias amigo. Tenemos trabajo pendiente por hacer en cuanto me quite esta "china" del zapato.

Por último y no menos importante, aunque no vayan a ser capaces de leerlo y sea extraño, a mis gatos, Mohoso y Houdini. Dios qué compañía te pueden hacer esos bichos cuando estás trabajando hasta las tantas de la madrugada, y los días que estas solo en casa. Yo creo que han escrito un 20% de la tesis de las veces que se han paseado por encima de mi teclado.

Que se puede decir del trabajo de cuatro años. Puede que por los tiempos que corren, tal vez debiera haber hecho un curso avanzado de instalación de aires acondicionados para facilitar mi inserción laboral, pero la realidad es que no lo cambiaba por nada. La he disfrutado mucho y, espero que la disfrutéis tanto de su lectura (si sois capaces de leerla) como yo haciendo este libro.

P.D: Si me olvido de alguien en los agradecimientos, soy un desastre. Si me conocéis, me disculparéis por ello y os lo agradezco también.

Gabriel G. Castañé.

Acknowledgements

This work has been partially funded under the grant TIN2013-41350-P of the Spanish Ministry of Economics and Competitiveness, the COST Action IC1305,"Network on Sustainable Ultrascale Computing (NESUS)", ESTuDIo (TIN2012-36812-C02-01), SICOMORo-CM (S2013/ICE-3006), the SEPE (Servicio Público de Empleo Estatal) commonly known as INEM, my entire savings, and part from my parents.

Resumen

En esta tésis se propone una nueva plataforma de simulación específicamente diseñada para modelar entornos de computación en la nube, sus arquitecturas subyacentes, y la energía consumida por los dispositivos hardware. Los modelos que consituyen los servidores se encuentran divididos en los cinco subsistemas básicos: sistema de procesamiento, sistema de memoria, sistema de almacenamiento, sistema de red, y fuente de alimentación. Cada uno de estos subsistemas ha sido modelado incluyendo nuevas estrategias para simular su consumo energético. Sobre estos modelos se despliegan los modelos de virtualización con la finalidad de simular el hipervisor y sus políticas de planificación. Además, se ha realizado el model del gestor de la nube, la pieza central de la plataforma de simulación y responsable de la gestión de las políticas de aprovisionamiento de recursos. Su diseño ofrece interfaces a los investigadores, permitiendo realizar sus estudios sobre políticas de planificación en entornos de computación en la nube.

Los objetivos de esta plataforma de simulación son permitir el modelado de entornos existentes y nuevos diseños arquitectónicos de computación en la nube, con un entorno configurable que permita modificar valores de consumo energético de los distintos componentes. Las principales características de esta plataforma son su flexibilidad, permitiendo una amplia posibilidad de diseños; escalabilidad, para estudiar entornos con gran número de elementos; y proveer un buen compromiso entre la precisión de los resultados y su rendimiento.

Se ha realizado el proceso de validación de la plataforma de simulación mediante la comparaciń de resultados de experimentos realizados en entornos reales, con los resultados de simulación obtenidos de modelar dichos entornos reales. Tras ello, se ha realizado una evaluación mostrando la capacidad de preveer el consumo energético de un entorno de computación en la nube que modela una aplicación real.

Finalmente, se han realizado experimentos para analizar la escalabilidad, con el fin de estudiar el comportamiento de la plataforma ante la simulación de entornos de gran escala. El principal objetivo de los test de escalabilidad consiste en calcular la cantidad de tiempo y de memoria necesarios para ejecutar simulaciones grandes, dependiendo de el tamaño del entorno simulado, y de la disponibilidad de recursos físicos para executarlas.

Abstract

In this thesis we propose a new simulation platform specifically designed for modelling cloud computing environments, its underlying architectures, and the energy consumed by hardware devices. The models that consists on servers are divided into the five basic subsystems: processing system, memory system, network system, storage system, and the power supply unit. Each one of these subsystems has been built including new strategies to simulate energy aware. On the top of these models, there have been deployed the virtualization models to simulate the hypervisor and its scheduling policies. In addition, the cloud manager, the core of the simulation platform, is responsible for the provisioning resources management policies. It design offers to researchers APIs, allowing to perform studies on scheduling policies of cloud computing systems.

This simulation platform is aimed to model existent and new designs of cloud computing architectures, with a customizable environment to configure the energy consumption of different components. The main characteristics of this platform are flexibility, allowing a wide possibility of designs; scalability to study large environments; and to provide a good compromise between accuracy and performance.

A validation process of the simulation platform has been reached by comparing results from real experiments, with results from simulation executions obtained by modelling the real experiments. Therefore, to evaluate the possibility to foresee the energy consumption of a real cloud environment, an experiment of deploying a model of a real application has been studied.

Finally, scalability experiments has been performed to study the behaviour of the simulation platform with large scale environments experiments. The main aim of scalability tests, is to calculate both, the amount of time and memory needed to execute large simulations, depending on the size of the environment simulated, and the availability of hardware resources to execute them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past decade, a new paradigm emerged from the scientific (and not scientific) community: cloud computing. Although the concept of cloud computing emerged in 2007, no formal definition exists yet [1]. Cloud computing can be seen as a paradigm that provides access to a flexible and on-demand computing infrastructure, allowing the user to request dynamically virtual machines to solve a computational problem. However, the views of different relevant actors in the industry appear to be irreconcilable concerning cloud computing.

In July 2008, the Cloud Computing Journal cited the attempts to define *Cloud Computing* of 21 independent experts, practitioners and academics [2]. A technical definition, according to Michael Armbrust et al. define the term Cloud Computing as "a new term for a long-held dream of computing as a utility, which has recently emerged as a commercial reality" [3]. Although there are currently several definitions for explaining what is cloud computing, the one provided by the U.S. National Institute of Standards and Technology seems to provide a precise definition including the key elements used in the cloud computing community [4]:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

At present, computing architectures based on cloud computing are probably the most cost-effective solution for final users: enterprises and scientists. As a result, this business model has been adopted by the great majority of important companies like Amazon, Google, Dell, IBM, HP and Microsoft. All of them have invested billions of dollars in order to provide their own cloud computing solutions. However, the market is expected to rise from $40.7 billion in 2011 to more than $241 billion in 2020 [5].

Migrating solutions to cloud computing environments has several advantages, such as being able to offer end-users a flexible computing system with the possibility of varying the number of CPUs and the memory size per job, even in runtime (if available). For the user, the ability to request on-demand resources, together with the lack of infrastructure

and administration results in ease of management and overall cost reductions. However, cloud computing is not an ideal paradigm. The huge size of the data centers that supports those systems requires managing resources efficiently, which entails the necessity of energy efficiency as well as energy proportionality. Moreover, due to the continuous rise of power consumption, a major cause of concern is the increasing power bills, carbon emissions, and power supply limitations for data centers.It is true that one data center is still better than the traditional disperse equivalent to scenario, but still efficiency is a problem to be improved.

The energy crisis of the last years, jointly with the ever increasing conscience about the negative effects of energy waste and climate change, have brought the sustainability both into public attention and under industry and scientific scrutiny. Green, or sustainable computing, has become the focus attention of initiatives such as Green Grid [6], a global consortium dedicated to advance energy efficiency in data centers and business computing ecosystems. This consortium has even established a metric called Power Usage Effectiveness (PUE) [7], which aims to measure and compare the energy efficiency of data centers. As demonstrated by the successful emergence of the Green500 list [8], which provides a ranking of the most energy-efficient supercomputers in the world, energy has become as significant as performance. Consequently, the performance-per-watt has been established as a new metric to evaluate supercomputers.

Currently there are several proposals for detecting, adjusting, and even reducing the main causes of energy consumption. Some of these are carbon footprint reduction [9], economical savings in IT electricity bills [10], and increasing the life time of some data center devices [11]. Thus, providing methods and techniques for analyzing the energy consumption in cloud computing systems becomes of vital importance. However, this is specially difficult and challenging because of the high number of users accessing the system concurrently and the wide variety of components that interact in the system, like communication networks, multi-core CPUs, users, virtual machines and hypervisors, that have a direct impact on both the overall system performance and energy consumption.

1.1 Motivation

Major requirements for cloud computing systems are scalability, elasticity, reliability and low-pricing [12]. However, providing a system with such interesting features entails different side-effects that have to be accordingly managed. One of the main issues currently faced by the cloud community is the energy required to maintain data centers, because they contain a vast number of computers and communication networks. Moreover, these systems are designed to be scalable, so that adding resources is a must in order to increase the overall system capacity, which addresses the ever growing need of energy supply.

Due to the fact that power consumption is one of the main issues in large scalable distributed systems, the efficient management of energy without loosing performance is currently a major concern in the field of cloud computing. In fact, there is a broad variety of proposals to save energy by maintaining a reasonable power-performance ratio. Current approaches for saving energy in cloud computing environments are widely different due to the difficulty of saving energy without end-users noticing a performance loss. On the one hand, it is necessary to perform a low-level consumption study of components such as

CPUs, memories, disks, and networks. The objective is to analyze the impact of different techniques in power consumption. On the other hand, it is necessary to study the behavior of techniques that shut down systems while trying not to penalize customers and users.

To achieve these tasks, the problem of capturing power consumption data for each node and for individual components must be faced. Depending on which technique is used, one may be interested in the aggregate power consumption of each node, or the real-time power consumption of the specific components in every node. In order to acquire power consumption data, several methods exist with varying level of precision, intrusiveness, and cost. The first consists in using power meter devices, which measure the aggregated power use of a machine. These devices are fairly unobtrusive because no hardware modifications are required to measure power usage. However, scaling this approach to large-scale systems can prove to be impractical due to the increased cost in hardware and logistics, since gathered data for every node needs to be stored and processed, which creates a big data problem. In addition, these power meters do not provide fine-grained measurements or component-level power usage data.

Another method involves directly instrumenting the motherboard with multimeters in order to obtain each power connector's voltage and current, thus obtaining real-time power consumption [13]. While this method produces precise, fine-grained results, it becomes impractical for production environments, because every system would require to be modified. The intrusiveness of these methods entails that in deployment is usually too aggressive for the expensive equipment that constitute the data center. In fact, they quickly become unmanageable, unfordable, and highly impractical when trying to scale this method to even medium-scale systems.

Hence, in order to analyze the trade-offs between performance and energy consumption in real cloud systems, one of these previous methods has to be used while executing a set of experiments in the cloud. Unfortunately, due to the associated cost, of using additional equipment and the intrusiveness required to analyze energy consumption in hardware-based cloud systems, this strategy is unfeasible. In addition, it is difficult to ensure the real capabilities of cloud computing architectures, or an improvement over an existing one, until the environment is completely deployed.

These difficulties can be alleviated by using simulation techniques for predicting the consumption of different hardware components in cloud architectures. In this case, the same experiments are performed in a virtual environment that represents the behavior of the real cloud system, that is, a simulated scenario. Basically, simulation uses existing power consumption models without modifying the hardware device. Both methods, using a real cloud with additional equipment for measuring energy consumption and simulation, have their own advantages and disadvantages. Some of them are described below:

- Simulation is cheaper than performing experiments directly in a real cloud. Renting machines from a public cloud requires a monetary investment, while executing a simulation only requires a regular computer.

- The flexibility obtained by using simulation is much higher than the one using real cloud systems. While in real clouds the users have to deal with the specific configuration of the system, simulation allows the users to quickly set up a wide range of configurations. These configurations may involve network topology, cost policies,

hypervisors, etc. For example, using simulation, users are able to design and test different resource provisioning policies in order to balance the trade-offs between performance and energy consumption. In the case of real clouds, users are not allowed to configure the resource provisioning policies of the system.

- Simulation models can be shared with other researchers of the community. Since the used simulation platforms are open source, they can be shared and freely modified to the own purpose of researchers.
- Capturing real-time consumption of different hardware devices is a costly process. The acquisition process of consumption values in real hardware-based systems consists on modifying the devices to allow the energy measurements. Moreover, it requires specific equipment. This problem is alleviated by using consumption models in simulated environments.
- In many cases, simulation experiments require more time to be executed than hardware-based tests. This problem can be alleviated by increasing hardware resources for the simulation by using parallel simulation techniques.
- Results obtained from executions over hardware-based systems do not need to be validated to ensure their accuracy. In a simulated environment it is necessary to validate the results.
- Simulation allows to simulate last still nonexistent solutions, to design new clouds or enhancing existing ones.

Hence, providing a simulation platform that supports to perform accurate studies of power consumption in cloud computing systems is challenging.

1.2 Thesis statements

- Simulating cloud computing architectures with enough level detail provide a good level of fidelity with its analogous hardware-based system. This requires that each part has to be represented like models of hardware devices, energy consumption models for these devices, and software models responsible for the resource management, imitating the correctly behavior of the target cloud to be simulated.
- Providing a method to simulate realistic workloads of cloud systems allows to foresee the energy consumed by the hardware of data centers. A workload in a cloud consists of a set of users that request resources for executing applications. This is specially challenging due to the large amount of users that access to the system and request resources in parallel.

1.3 Main objectives

This thesis addresses the challenges previously commented. Its main objective is to **provide new contributions to model and simulate energy-aware cloud computing**

systems. Hence, in order to fulfill this objective we propose to satisfy the following more specific goals:

1. **Designing strategies for modelling and simulating the underlying architecture of cloud systems**. These strategies have to deal with different challenges. First, each hardware device that conforms the cloud must be modeled by accomplishing a good compromise between performance and fidelity with its analogous hardware device. Second, the underlying architecture of the simulated cloud system must represent the architecture of the target cloud system. Third, the modeled cloud systems must provide scalability independent of the size of the cloud to be analyzed.

2. **Designing non-invasive strategies for simulating the power consumption of cloud systems**. These strategies must be designed for analyzing, measuring and managing the energy consumption in cloud systems. Moreover, this approach must fulfill the following requirements: the analysis of energy consumption in cloud systems must not require an additional equipment; performing energy consumption experiments must be non-intrusive, that is, it is not required to access those hardware devices that are modeled in the simulated cloud. Finally, this approach must be customizable by users, that is, the proposed strategies must be easily modified by users in order to allow the study of many scenarios.

3. **Analyzing the impact of realistic workloads on the overall system energy consumption in different simulated cloud systems**. In order to perform a wide variety of realistic experiments, different application models must be simulated. Basically these experiments include different realistic application models, the costs and performance to execute them depending on users purchased resources, and the impact on the energy consumption of the system by using different configurations. Moreover, users must be simulated as owners of the applications, being distributed along time for accessing the cloud, launching applications with their own task scheduling policy over the purchased resources.

This thesis is focused on evaluating different workloads and its the impact on energy efficiency over cloud computing systems. Using this approximation, we can analyze the trend of energy consumption of the entire system using non-invasive strategies.

1.4 Structure of this document

The rest of this document is organized in six chapters whose contents are summarized in the following paragraphs:

- Chapter 2, State of the art, is a review of current works about simulation and energy aware cloud computing environments. This chapter describes the most relevant concepts for power management, the consumption by components and energy models and the techniques to save energy in cloud computing environments. Next, the most relevant simulation tools for modelling cloud systems are summarized.

- Chapter 3, Modelling and simulating the underlying architecture for cloud computing systems, shows a proposal for a flexible, scalable and expandable simulation platform

for modelling and simulating energy aware cloud systems. The main objective of this chapter is to design strategies for modelling and simulating different architectures of cloud systems, covering goal 1.

- Chapter 4, Modelling and simulating virtualisation of hardware resources, describes the process for managing the virtual resources of cloud systems in a simulated scenario. In this case, a module called hypervisor is in charge of managing the requests of physical resources through virtual machines. Moreover, some algorithms for managing the resource provisioning are presented using the simulation platform designed in Chapter 3. This chapter covers the objective 2.

- Chapter 5, Case Study, shows experiments for analyzing the impact of different workloads in different cloud architectures. In order to analyze the performance and energy consumption of each system, we need to model different applications workloads, that is, how users accesses to the cloud system requesting resources, how the performance of applications vary depending on purchased resources, and how applications executions impact on the energy consumed by the data center. The full list of goals are reached at this point by covering the objective 3.

- Chapter 6, Conclusions and future works, presents the conclusions of this work and also describes some future works.

  Finally, the bibliography used during the elaboration of this work is provided.

Chapter 2

State of the art

This chapter exhibits the state-of-the-art of modelling and simulating cloud computing environments. Specifically, it is focused on energy-efficient techniques and simulation frameworks, providing a classification of energy efficient solutions. The chapter finalises with a brief description of the most relevant modelling and simulation tools in the scope of cloud computing.

2.1 Cloud computing

In recent years, cloud computing systems are increasing their role due to the fast increase of computer networks, communication technologies and the continuous price drop of computer devices. A very clear proof of this fact is that important companies are investing billions of dollars in order to provide their own cloud solutions [14]. The market for cloud computing services estimates that the software as a service (SaaS) generated revenues of $11.7billions/year, platform as a service (PaaS) $311 billion/year and infrastructure as a service (IaaS) $1 billion/year in 2010. However, it is expected to rise $52 billions/year in 2020 to PaaS/SaaS market and $4 billions/year to IaaS market [15].

For the sake of clarity, the definition of cloud computing is provided by taking into account two different perspectives: users and provider. First, from the end-users point of view, cloud computing can be seen as a technology that gives the illusion of providing access to infinite resources that are dynamically adapted to the user's needs. Hence, users can configure powerful systems depending on both the budget and requirements of computing power, memory and storage. From the cloud providers point of view, cloud computing is a business-model based on renting virtual resources to users by establishing a cost depending of the quality of these resources and the time-frame that users have access to them. Generally, these resources are abstracted by using Virtual Machines (in short, VMs).

The term virtual machine is defined by Popek and Goldberg in 1974 as "an efficient, isolated duplicate of a real machine" [16]. At present, the use of VMs has evolved. Smith and Nair include in the definition of VM the possibility of "supporting individual processes or a complete system depending on the abstraction level where virtualization occurs" [17]. This definition is closer to the existing virtualization in cloud computing environments.

Cloud computing covers a wide spectrum of areas, including research community,

IT industry and entertainment companies. The main (ideal) motivation for using this technology is the need by end-users to deploy their applications using a unsettled set of resources, isolating themselves from infrastructure of the data centers, without maintaining the system and paying only the resources that they use. Moreover, a very attractive feature of cloud systems is the possibility of scaling convenient resources [18].

2.1.1 Cloud computing models

The ever-increasing computing power demands for large-scale applications entail the increment of maintenance costs. The cloud computing community has been deploying modern computing systems by offering flexible ad-hoc systems in size and resources. Basically, these cloud systems can be classified in two categories: cloud deployment model and cloud service model. The former one focuses on the business model of the data center that hosts the cloud computing environment, while the second one classifies cloud systems according to the IT offers of infrastructures and network environments.

According to the cloud deployment model, each cloud system can be categorised in [19]: public, private, hybrid, science and community cloud. The term known as *public cloud* defines a business model where the cloud provider makes applications, storage and other resources available over Internet, in a pay-as-you-go manner, to the general public. Instead of made available to the general public, if the system is a single private data center used, managed, and hosted by an organisation, it is called *private cloud*. With a composition of both described before, if users are individuals and companies, a cloud is known as *hybrid cloud*. Other deployment model is the *science cloud* [20]. The concept of science cloud appears in 2008 as a project between Florida and Chicago Universities. K. Keahey started this project that "allows members of the scientific community to lease resources for short amounts of time". A science cloud do not use the model pay-as-you-go, neither does require that users pay for usage like private clouds. It is only verified that the person requesting resources is indeed a member of the scientific community. Finally, the last deployment model is called *community cloud*, where consumers have common concerns, such as policies, goals and security requirements, and several organisations shares the infrastructure in order to save costs.

The cloud service model consists of three categories [12]: Software as a Service (in short, SaaS), Platform as a Service (in short, PaaS) and Infrastructure as a Service (in short, IaaS).

SaaS defines the set of applications - commonly web based applications - given by providers which run on a cloud computing infrastructure. In this layer, the infrastructure is transparent to the end-user, limiting users to the management of the application; only a few configuration settings are permitted from a client interface. On the contrary, PaaS and IaaS are less focused on users. In the PaaS model, consumers can develop onto the cloud infrastructure their own applications by using different tools (program languages, libraries, and services) supported by the cloud provider. The infrastructure is transparent like SaaS model with the exception that, in the PaaS model, consumers are able to manage the configuration settings for the application-hosting environment. PaaS applications are deployed by users, which usually offer those services to thirds. IaaS layer provides to end-users the capability of managing the storage system, the processing system, the operating system and applications. The cloud provider is in charge of managing the cloud infrastructure

where, through virtualisation techniques, users receive VMs assigned to physical machines of the data center. On request, once a VM is booted on a physical host, it can be used like a dedicated physical machine. The cost of renting this VM is assigned depending of the quality of the requested resources (number of CPUs, amount of RAM, etc.).

At present, there is a trend that consists in building acronyms with the suffix "aaS" to whatever that can be sold by IT enterprises as services. Some examples of new appearing layers in the cloud service model are HaaS [21], PraaS[22] and DaaS[23]. HaaS is the acronym for Hardware as a Service and it refers to users that use IT hardware - or even an entire data-center/computer center - as a pay-as-you-go subscription service. The HaaS model can be flexible, scalable and manageable to reach their needs. PraaS stands for providing the management of a complex process in the cloud. DaaS, Data as a Service is the management of data in various formats from various sources. They can be accessed via services to users on the network. Users can, for example, manipulate remote data just like operate on local disk; or access data in a semantic way on the Internet.

2.1.2 Virtualisation techniques in cloud computing

The most popular technique to virtualise resources lies in a software which allows multiple operating systems instances (guests) over a host computer. This software - virtual machine monitor - was described in 1973 by R. Goldberg [24]. Currently, in cloud computing environments, the virtual machine monitor is named virtual machine manager or hypervisor.

The hypervisor takes control over the VMs. A VM does not have access to the physical processor, nor does it handle its real interrupts. Instead, it has a virtual view of the processor and runs in guest virtual address. The hypervisor handles the interrupts to the processor and redirects them to the respective partition. According to Goldberg's classification, hypervisors can be categorised in hosted and native/bare metal:

- Hosted hypervisors run within an operative system. This operative system is still visible and usable by user. The main scheme consists of three layers: the native operative system, the hypervisor, and the executed VMs. Some examples of this type of hypervisors are VMWare Workstation [25], VirtualBox [26], Parallels Workstation [27], and Qemu [28].
- Native hypervisors run with exclusive accesses on the host machine. The access management from VMs to hardware resources is responsibility of the hypervisor. This is the case of XenServer [29], KVM [30], VMWare ESX/ESXi [31] (Elastic Sky X), and Hyper-V [32], that are some of the most important virtualization vendors in cloud computing.

Virtualisation techniques for getting the isolation of the virtual machine in computing systems are classified in three categories, where each category refers to the responsibility of hypervisor module when VMs invoke calls to hardware resources. These three categories are:

- Full virtualisation with binary translation. The main target of a full virtualised hypervisor is the simulation of privileged operations as I/O instructions. Virtual operations

are not allowed to alter the control program or the hardware. Some machine instructions can be executed directly on the hardware but controlled and managed by the hypervisor. Full virtualisation does not change the guest operative system. This kind of virtualisation allows the use of generic device drivers.

- Hardware assisted virtualisation. Virtualisation services were deployed on supported hardware without additional hardware. The virtual machine monitor is integrated with the hardware, leaving the Guest OS unchanged and allowing the use generic device drivers.

- OS assisted virtualisation - Paravirtualisation. This model allows the guest operating system to have knowledge about the tasks that the host is executing and vice versa. The guest operating system, and its drivers, require to be adapted, that is, paravirtualised. The paravirtualised scheme decreases the performance reduction, commonly given by the execution of VMs inside the virtual-guests, more than other virtualisation techniques. In this case, the hypervisor is a thin layer which guest access to host hardware.

2.1.3 Benefits and drawbacks of cloud computing

Enterprises, governments and research institutes have been striving to boost cloud computing environments [12]. The benefits to adopt this paradigm are listed below:

- Self-service on demand. As it is defined by the National Institute of Standards and Technology [19], self-service is the process through "a user can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider". This process, that may be viewed as a type of SaaS, is the capability by users to request the resources that they want when they need.

- Accessibility. Cloud computing capabilities are available on Internet by network. Due to the power of mobile phones, tablets and laptops, this access is promoted by using heterogeneous platforms, and may be performed by different standard mechanisms. This is due to the access is over the network through standard mechanisms.

- Elasticity. A cloud computing environment has the capacity of scale on demand. Users can increment or decrement the resources as they need. This capability for elastic provisioning, from the consumer's point of view, appears to be an unlimited resources solutions for their needs.

- Resources pooling. The resources of data centers are pooled to be served to multiple users. Those users may request different physical and virtual resources, dynamically assigned according to consumers demand, such as storage, processing, memory, network bandwidth, virtual machines, and email services, among others. This resources pooling allow to cloud providers the control of the resources that users are consuming.

- Pay-as-you-go. The resources usage is measured and controlled, providing transparency for both providers and consumers. Cloud computing services use a metering

capability which enables control over the use of resources. This implies that IT services are charged per usage metrics, that is known as pay per use or pay-as-you-go business model. In general, consumers and enterprises can avoid the following costs for maintaining the infrastructures, recruiting qualified staff to manage the data center, paying energy bills and the obsolescence of the hardware.

- Carbon footprint reduction. Virtualisation techniques allow to save energy from data centers. These data centers offer the possibility to reduce the number of physical servers by using several methods like server consolidation. It consists on switch off the nodes that are not needed.

However, cloud computing is not an ideal paradigm. Generally, this paradigm is built over clusters where the inherited drawbacks of these systems are not easy to avoid. Distributed system architectures is a field with decades of experience that clearly demonstrates the critical factors of connection latency and bandwidth, the general consumption of the nodes and the tolerance to hardware failures. To these inherited drawbacks, new ones given by the underlying architecture of cloud systems must be added. Following, a summary of these drawbacks are presented:

- Security. This is an important obstacle that concerns the cloud computing paradigm. Some of the top cloud computing threats identified by the Cloud Security Alliance (in short, CSA) [33] are the extraction of private cryptographic keys using the side-channel timing information from a virtual machine; the unauthorised access to user's credentials, manipulation of data, falsification the information or redirection of clients to illegitimate sites by traffic hijacking; and the typical denial of service (in short, DoS) attack, when a server is blocked due to the excessive number of requests.

- Privacy. Ensuring the privacy of user's data is important and complex to reach by cloud services. Therefore, the cloud provider is responsible for managing the access to the data by forbidding unauthorised users, causing users to be uncomfortable by trusting their sensitive data to a third party. Furthermore, not only this dissatisfaction by users or corporations, but also requirements and regulations must be provided in order to move corporate data to a cloud computing environment.

- Offline. Applications and data are limited to on-line support. A failure by an Internet provider can cause the lost of "the high-availability computing community" due to the lack of accessibility of data and applications allocated in the cloud provider.

- Dependency. Users are not only dependent of their ISPs to maintain their cloud services, in addition some Cloud Service Providers (in short, CSP) have quality problems which cause user-CSP dependancy. Some of those CSPs dependency problems arise by the following reasons: the cloud system do not provide tools to migrate to another CSP, inexistent or uncompleted tools to backup and lack of restore or recovery procedures.

- Performance unpredictability. The virtualisation layer hides the network links and the physical state. It is very common that several VMs of different users are allocated over the same physical node. These VMs request resources varying the performance of the applications and decreasing the response times and data capacity of the network.

M.Ambrust reflects that "the performance unpredictability in the cloud is in fact a major issue for many users and it is considered as one of the major obstacles for cloud computing" [3]. J.Schad also shows that there are several open issues - such as performance variance on the cloud and static scheduling of MapReduce jobs - that are not addressed yet and have a negative impact on performance of applications [34].

- Increment of energy consumption. Currently, an important drawback in cloud computing systems has focused on the ever increasing energy consumption of its data centers and their growing electricity bills by cloud providers side. In spite of a better use of the equipment by using virtualisation techniques, the energy consumption is focused in single areas, where the data centers are allocated. This entails the problem of supply enough energy for the power consumed by nodes, and the power consumed for refrigerating the room where nodes are running.

2.2 Energy-efficiency in computing systems

Currently, a high number of initiatives in both scientific community and industry have a common aim: an energy consumption decrement in terms to do more operations per watt in computer systems. Important IT companies such as AMD, HP, IBM, DELL, INTEL, Sun Microsystems, VMWare and Microsoft are researching in energy-efficiency of data centers to reduce the usage costs.

L. Barroso predicts that in the future *"if the power consumption per server continues increasing, the cost of the energy consumed by a server during its lifetime will exceed the hardware costs"* [35].

The scope of energy-efficiency in computer systems can be categorised in energy background, power consumption by single components and energy models and power management techniques.

2.2.1 Energy background

The definition of power consumption might be blurry depending on the field of science that defines it: chemistry, physics, biology, astronomy, engineering, etc. Before describing energy models and energy saving techniques in cloud computing environments, this section focuses in the differences between the power/energy consumption, and the static/dynamic power consumption in the context of computer science, which are the basis of several energy-efficient policies.

V. Venkatachalam et al. suggest that power and energy are commonly defined in terms of the work that a system performs [36]. Energy is the ability of an electrical device to perform a work, while power is the rate at which a system performs that work. In other words, how fast the energy is consumed. For example, lowering the CPU performance while a program is executing involves a minor power consumption, but it does not guarantee a reduction of the consumed energy. If that program needs more time to finish its execution, the energy consumed by the CPU will be the same. This means that a reduction of the power consumption not always entails a decrement in energy consumption.

The main consumption of a circuit is proportional to the flow of electric charge (Am-

peres) transferred by it and the voltage given by power supply. A decrement in power consumption entails a minor heat and, consequently, a major duration (life-time) of components. The total consumption of a circuit consists of static power consumption and dynamic power consumption.

The static power consumption depends on the state of the circuit, the volts given by power supply, and details of the fabrication processes. A reduction of power consumption in this scope entails hardware modifications in design processes [37].

The dynamic power consumption arises from circuit activity. It has two sources: short-circuit current and switched capacitance. Short-circuit causes the 10%-15% of the power consumption. It is produced in transistors during the transitions from active/inactive state to the opposite one. The transistors are, in the short period of time when a transition occurs, in short-circuit state. Due to the high number of transistors into a microprocessor (more than one billion), and the high number of times that a transition is performed, the consumption increases [38]. It is difficult to reduce this energy losses without comprising the performance.

Switched capacitance is the consumption produced by charging/discharging the circuit capacitors. There are two possibilities to decrease the switched capacitance consumption. A reduction in the size of the transistors, dropping the performance, is the first possibility. The decrement of switching activity is the second one. Switching activity can be reduced by using the Dynamic Power Management (DPM) technique.

2.2.2 Power consumption in computer devices and energy models

Many components have a great influence on the power consumption of a datacenter. In order to aim the analysis, the power consumption will be breakdown by following devices: CPU, memory, power supply unit (PSU), storage, network, and cooling systems.

CPU system

Currently, the main part of the dynamic power range and power consumption in a server is due to the CPU. There is a wide range of studies about its consumption.

L. Minas and B. Ellison, analysed the capabilities of servers and their power consumption [39]. The main part of power consumed by a server is due to the CPU but it does not dominate the total power consumption by a server. Newer processors support power saving states being much more power efficient than previous generations. An Intel Xeon Processor E5-5670 family can reach a peak performance of 146 GFlops using less than 10 kW of power [40]. Pentium processors in 1998 would have consume about 800 kW to achieve the same performance.

The power consumption in CPUs is different depending on the type of processor. W. Bircher and J. Lizy have performed a detailed study of power and performance on AMD quad core Opteron and Phenom processors [41]. They measure power by using a serie resistor, sampling the voltage across the resistor at 1KHz. The work takes a very close look at power on two processor cores and off-core resources, but they lack of other manufacturer processors and different micro architectures.

H. Esmaeilzadeh et al. report and analyse measured chip power and performance on

five process technology generations [42]. They measure eight Intel processors with technologies from 130nm to 32nm introduced in 2003 through 2010 by executing 61 secuential and parallel benchmarks in different languages. They explore the effects of simultaneous multithreading (SMT), core count (CMP), clock frequency, die shrink, gross microarchitectural changes, Turbo Boost, software parallelism and workload.

X. Fan et al. detail the relationship between CPU utilisation and total power consumption by a server [43]. The idea is to use a non-linear model where is the power consumption by an idle server, is the maximum power consumption by the server, u shows the CPU utilisation reported by the OS as an average across all CPUs and r is the calibration parameter. This calibration parameter minimises the squared error. Authors recommend for each class of machines deployed one set of calibration experiments to produce the following model:

(2.1)

L. Minas and B. Ellison simplified this model [39] to:

— (2.2)

An estimation of power consumption (P) at any specific processor utilisation (n) can be calculated if power consumption at maximum performance ( ) and at idle ( ) are known.

SDRAM system

The second part of power consumed by a server is drawn by the memory [39]. Memory chips consumption are increasing, varying from 5W to 21W per DIMM in DDR3 and FB-DIMM memories [44].

K. Chandrasekar et al. improved a power model to estimates the power consumption that estimates power consumption during the state transitions to power-saving states [45]. The procedure used to obtain the data is generic and applicable to all DDRX SDRAMs, all memory controller policies and all degrees of bank interleaving. The data are obtained by employing a SDRAM command trace to get the timings between the commands issued.

Generally, the memory basic commands are pre-charge (PRE), activate (ACT), read (RD), write (WR) and refresh (REF) [46, 47]. In addition to these commands, if the memory is not in use, the transition to power-down state by disabling the clock at run-time is also possible, achieving a reduction in power consumption. It is also possible to retain the memory contents in the power-down state by employing the Self-Refresh feature, to refresh the memory at significantly lower power consumption.

Hence, a feasible method for estimating SDRAM power consumption consist in using the current, voltage and timing constraints values from the SDRAM data sheets, which are based on real measurements. The relationship between SDRAM utilisation and power consumption is highly detailed and it consists of 20 formulas to model the states and transitions between them [45].

The growth in frequency and capacity parameters of actual memory chips entail several problems in power consumption, cooling systems and performance. L. Minas and B.

Ellison studies this problem [39]. A DIMM under double refresh has a case temperature specification of ° rather than ° , thereby enabling a higher overall safe system temperature at the expense of a small power loss, and slightly increased power consumption. The impact of double refresh (° versus °) is improving cooling capability by approximately two to three watts, resulting in a significant improvement in memory bandwidth capability.

Power supply unit system

The third part of power consumption by a server is drawn by the PSU. It is responsible of transforming alternate current (AC) to low voltage regulated - direct current (DC). During this transformation, a significant power is lost in heat. Efficient PSU's requires less airflow to its cooling efficiency and also the energy lost as heat is minor. In 2005 computer power supplies had an average of 70%-80% [48]. A list of all certified power supplies and manufacturers is presented in a portal called *80 plus* [49]. Therefore, it is possible to confirm that in 2013 some PSUs exceeded 90% efficiency at medium load. For example, HP server power supplies can reach 94% efficiency, loosing efficiency at worst load levels, falling to 87%-89%. However, it is possible to see that those values vary from a model to other. It is essential for data centers to have constant power supply. Efficient power supply units save electricity and emit less heat by wasting less power.

Storage system

The amount of data generated and needed by users is growing continuously. This increment in the storage needs is often solved by adding more Hard Disk Drives (in short, HDD), whereas these drives require an additional power. Generally, HDDs states are Idle, SATA or SCSI Bus Transfer, Read, Write, Seek, Quiet Seek (additionally, if supported), and Start.

There are several alternatives to model HDD consumption, like the approach described by D. Molaro et al. [50]. The authors of this work proposed a method to estimate the disk power consumption using a real-time streaming workload trace of all operations requested to the disk. Their main assumption is to treat separately the data transfer and seek component of a disk request.

L. Minas et al. analyse the power consumption and storage models [39]. The average power consumption of a HDD can be calculated by measuring the power consumption of a HDD both during typical user operations and during intensive (constant) operations. For every usage model, the idle versus active percentage of HDDs depend of disk capacity, applications in use and related workloads. Average power consumption formula is an estimation that may vary depending on the HDD state duration. The formulas are based on the assumption that read/write HDD operations constitute the 10% of the total time for the average office usage and over 50% of the time for the intensive operations usage. However, it does not contemplate more than 50% data-intensive writing and reading.

A. Hylick et al. instrumented at a fine-grained level a set of ten HDDs varying its age, capacity and interface [51]. The results show that the energy consumed by the electronics of a drive is just as important as the mechanical energy consumption. The energy required to access data is affected by physical location on a drive; and the size of data transfers has measurable effect on power consumption. It exists disparity between reading/writing

operations, the energy consumed varying the logical block numbers (LBNs), and the energy consumed of same-size transfers with different block sizes. The energy consumed during seeking is minimal, but restricting disk accesses between low and central LBNs could aggregate into savings over extended usage periods.

The relationship between hard disk utilisation and total power consumption by a server using performance data was also presented by A. Hylick et al. [52]. The authors of this work relate that the trade estimator of the energy consumed by a HDD can be obtained as a composition of active energy (read/write operations), seek energy and idle energy. The trade estimator of the total amount of energy consumed by a drive servicing a set of N requests, including I (time idle), and comprised of S seeks, may be modelled as:

(2.3)

Where is the total energy, in Joules; given by A. Hylick et al. and represented by a formula which considers that seek time increases when distance between the accessed data increases too; is obtained from the data-sheet provided by the manufacturers of HDD; is obtained by knowing the approximate bandwidth at a given LBN. The energy consumption for data transfers beginning at that LBN is a function of the time spent transferring the data.

The most efficient HDDs will consume in average 5W - 6W in idle state. In average, HDDs with SATA interface consume between 7W and 10W in idle state, and between 10W and 15W during active modes [39].

Network system

Cloud computing environments can consume more energy than conventional environments due to the transmission and switching networks that connect users to cloud services. The growth on the use of network produces a researching area which is based on studying, modelling and saving energy of networks and communication systems.

J. Chabarek et al. examine the problem of power-awareness in wire-line networks [53]. They build an energy consumption model to routers. The main conclusion of this work is that it is best to minimise the number of chassis that are powered at a given points of presence, maximising the number of line cards per chasis. All the experiments are realised over Cisco routers (Cisco GSR 12008 and Cisco 7507) [54]. Otherwise they don't perform experiments over other well-known vendors as i.e. ProCurve [55] or Brocade [56].

P. Mahadevan et al. describe the hurdles in network power instrumentation by performing studies of networking elements such as hubs, edge switches, core switches, routers and wireless access points in stand-alone mode and a production datacenter [57]. They build a benchmarking suite that allows them to achieve a consumption model. The main conclusions showed by Mahadevan are the existence of a great variability amongst switches with respect to their maximum rated power; the energy consumed by a switch increases linearly with the number of line cards plugged into the switch; the energy consumed by a switch is largely independent of the packet size for a fixed traffic throughput; and ideally, devices should consume energy proportional to their load, defining an energy proportionality index (EPI) as measurement unit for network devices [58].

J. Baliga et al. present a detailed energy consumption analysis of cloud computing [59]. This analysis considers the energy consumption in switching, transmission, data processing and data storage of both public and private clouds. Public cloud storage consumes the order of three to four times more power than private one. The reason is the increased energy consumption during the transport. As the number of file downloaded increases, the energy consumption used during the transport and storage grows. The model of energy required to transport one bit from a data center to a user at a corporate network ( ) is defined by:

—— —— — (2.4)

where , , and are the capacities of the corresponding equipment in bits per second respectively. , , and are the powers consumed by the Ethernet switches, small Ethernet switches, and data center gateway routers respectively. The factors of three accounts are, for the power requirements for redundancy (cooling and overheads) - factor of 2 x 1.5- and the factor of 3 due to the under utilisation of a corporate LAN. The authors of this work assume that a private cloud would significantly increase network traffic. Therefore, in their model, they assume an average utilisation of 33% (less than 5% without using a cloud environment [60]).

The model required to transport one bit from a data center to a user through the internet ( ) is defined by:

—— —— — —— ——— —— (2.5)

where , , , , , , and are the powers consumed by the Ethernet switches, broadband gateway routers, data center gateway routers, provider edge routers, core routers, and WDM transport equipment, respectively. , , , , , , and are the capacities of the corresponding equipment in bits per second. Power consumption and capacities of equipments are given by manufacturers at data sheets.

There are other research studies in energy aware networks. Y. Shang et al. make a comparison study on the energy proportionality of data center networks, based on regular routing and energy-aware routing [61]. Moreover recent researches are focused in two representative methods: device-level sleeping technology [62, 63] and network-wide energy aware routing [64].

Cooling system

Usually, data centers are designed to hold maximum number of computers at maximum operation capacity. The size of them may vary from a small room to large buildings hosting racks of computers. Each rack is connected to an uninterruptible power supply (in short UPS) which acts as a battery backup to prevent energy failures. These data centers operate 365 days a year, 24 hours, where servers and UPS generate huge amounts of heat. This heat has to be suppressed by computer room air conditioning units which consists of fans, cooling coils and air compressors. S. Greenberg shows in the results of benchmarking 22 data centers that about 30-40% of electricity consumption goes to the cooling equipment [65].

The analysis of cooling systems effectiveness for real time data center is unfeasible. The data center IT load can vary from an instant to the next, as it is showed by M. Seymour et al [66]. The authors of this work state that it is difficult to capture the IT cooling and configurations with desired accuracy. Z. Rongliang et al. have developed thermal models to data center cooling management and analysis [67, 68]. They perform physics base state-space models describing the air flow transport and distribution within the data centers. The parameters used in the models are obtained from measured data of system identification experiments and hence, they are ensured to reflect the data center reality emphasised.

Furthermore, there are alternatives to improve the cooling system consumption, such as using water to refrigerate the data servers. I. Madhusudan et al. developed a cooling system which comprises of 100% liquid cooled server cabinet, and an outdoor dry cooler unit for heat rejection to the ambient [69]. This configuration is used to reject the Information Technology (IT) equipment heat load directly to the outside ambient air without the use of a water-cooling machine. Using this technique they reach an energy saving of 25% at the tested datacenter.

Several important companies and researching centers starts initiatives to mitigate their heat problems. Some of those alternatives consists of allocate their datacenters in colder geographic areas or emplacements. This is the case of Google, which allocate their servers in Helsinki (Finland), using water of the Baltic sea to the cooling system [70], or Barcelona Supercomputing Center, which uses an old abbey to emplace the supercomputer Mare Nostrum [71].

2.2.3 Power management techniques

Power management techniques were born as a method to enlarge battery life of portable devices - embedded systems and laptops. L. Benini et al. defined Dynamic Power Management (in short, DPM) in 1999 as a design methodology that dynamically reconfigures an electronic system to provide the requested services and performance levels with a minimum number of active components, or a minimum load on such components [72]. They relate the fundamental premises for the applicability of DPM in systems (and components).

Depending on where DPM techniques are used, they can be broadly divided into several categories. However, focusing DPM in cloud computing, specifically in the layers of a node, these can be classified in two categories: hardware-firmware level and virtualisation level.

Hardware-firmware level

L. Barroso and U. Holtze relate that many techniques developed for mobile devices showed a direct benefit on general-purpose servers [58]. The major part of these techniques manage hardware resources to save energy. However, hardware transitions from a state to other entail penalties. The transition from low power or deactivated state to high power state, involves an additional power consumption. From high power to low power state, it entails the re-initialisation of components, by comprising delays which not always can be assumed by other components, or the applications that could be executing.

The application of DPM on hardware-firmware level consists of of two main techniques: Power State Machine (PSM) and Dynamic Performance Scaling (DPS).

L. Benini et al. model power-managed systems as a set of interacting power manageable components controlled by a power manager [72]. The technique for energy management is represented by a state machine named as Power State Machine (PSM). This simple abstract model holds for many single-chip components like processors and memories, as well as for devices like disk drives, wireless network interfaces, and displays which are more heterogeneous and complex than a single chip. PSM is the core of serveral studies [73, 74].

With the advent of multiprocessors new strategies appeared. M. Curtis et al. proposed different approaches for energy saving, namely processor deactivation and execution contexts, based also in PSM techniques [75]. They use physical experimentation on a real simultaneous multithreading system (in short, SMT) and a power estimation model. The authors of this work are based on the die areas of processor components, and component activity factors obtained from a hardware event monitor. The processor deactivation always ends up in power savings, whereas thread deactivation within active processors may or may not result in power savings. Power consumption is linear to the number of active processors. Depending on the processor architecture, threads deactivation within a processor reduces power (or not). On multicore processors, thread deactivation will always result in power savings, since entire cores, each occupying significant area of the die, will be powered down. On SMT processors, which share a common pipeline between multiple threads, power savings are not necessarily linear to the number of deactivated threads, since the remaining active threads may still occupy the released resources to maximise their own instruction-level parallelism (ILP). Deactivating threads on SMT entails a power reduction only if the load for critical thread-shared resources, such as the caches, is reduced.

CPU is not the only device that support the PSM technique. It is also supported by HDDs, but storage entails the problem of redundancy. Pinheiro et al. present a method that consists in placing data and parity on different disks, deactivating parity disks by during light loads and staging parity updates in non-volatile RAM [76].

PARAID [77, 78] and e-RAID [79] are works focused on modifying the disk architecture to increase their power efficiency. PARAID is a power-aware disk array architecture that exploits the unused storage space of active disks by replicating blocks from inactive disk. e-RAID usage transforms reads for RAID-1 and RAID-5. Hence, it allows system accesses to the contents of inactive disks from cache, other active disks, or both.

At present there are several algorithms to group sets of disks and switch on/off them depending on the predefined conditions to save energy. An example of this is ISRA (Immediate Successor Relationship Amount). The main idea of this work, developed by L. Xue-Liang et al., is to group the disks by frequent successive accesses [80]. They define the concept of Immediate Successor Relationship Amount (in short, ISRA) to represent the successor relationship of data blocks, based on an undirected graph. Data blocks that have frequent successive accesses are grouped and sorted by using a merge-sort-like algorithm to determine the position of every group, as well as the new position of every block within these groups. Their results show that both disk seek time and the energy needs could be reduced by about 50%.

The wide spectrum of techniques and algorithms for PSM technique over disks are summarised by T. Boston et al. in [81].

The second hardware-firmware level technique for energy saving is Dynamic Performance Scaling (in short, DPS). Motivated by emerging battery-operated application in 90s

decade, AP. Chandrakasan et al. presented the root of DPS, the low-power CMOS [82]. There are four major sources of power dissipation in CMOS circuits, which are summarised in the following formula of the dynamic power consumption ( ):

(2.6)

Where a is the switching activity, C is the physical capacitance, V is the supply voltage and f is the clock frequency. The changes in supply voltage or clock frequency parameters are the basis of Dynamic Voltage and Frequency Scaling (in short, DVFS). The main idea of DVFS is to decrease the voltage and frequency of the CPU, down-scale CPU performance when it do not affect to execute applications. In other words, the less voltage or frequency is used by CPU, the less power is needed to execute, but also the less performance is obtained.

At present, DVFS studies are divided into several researching areas. Some of the most relevant scopes of DVFS are the processor [83, 84], architecture [85, 86, 87] and algorithms [88, 89].

A specific type of DPM algorithm implemented in the hardware as a part of the electronic circuit was addressed in 1996 by Intel, Microsoft and Toshiba. They publisheed the first version of the Advanced Configuration and Power Interface (in short, ACPI) which attempt to unify the existing power/configuration standards for the hardware devices. The goal of ACPI is to provide an interface that can be used to apply DPM techniques into the operating system by adjusting the power states. ACPI standard define C-states (Processor states), P-states (performance states), D-states (Device states) and G-states (Global states), as the different working states of a processor. INTEL provides iASL, an ACPI Source Languaje Optimizing Compiler and Disassembler, which is a fully-featured translator for the ACPI Source Language (ASL), and ACPI binary data tables to the operating system [90].

Virtualization level

The highest level category of energy saving (virtualization level) appears with cloud computing environments. With the aim of saving-energy, VMs are migrated to other preselected nodes with the goal of switch off the unused servers. However, those techniques have an adverse effect on the performance. VMs migration is a process that consume time and resources.

Before cloud computing environments, process migration was a hot topic in systems research during 80s [91, 92, 93]. In 2003, A. Spellman et al. relate that consolidation efforts can provide significant savings in IT expenditures, like lower the total cost of ownership, improved service levels and availability, and reduced business risks, resulting from consistent business management and resilience [94]. Their work is focused on software scalability for analysing the performance impact of consolidate many servers onto a few servers by moving processes.

S.Srikantaiah et al. relate that with the advent of cloud computing environments, the grain of consolidation servers is the VM [95]. This is because in cloud computing approach multiple data center applications are hosted in a common set of servers. The goal of energy aware consolidation is to keep servers well utilised. For example, the idle power costs have

to be efficiently amortised, but without taking an energy penalty (performance degradation and power variation) due to internal contentions.

C. Clark et al. describe the process of moving transparently a VM from one physical host to another physical machine, while the VM is still powered on [96]. They named the process: "Live Migration of VMs". It has become a significant gear of load balancing algorithms and power management strategies to servers consolidation. Nowadays, virtual machine live migration is considered a default feature in hypervisors.

During the process of VMs live migration, four parts need to be considered: memory state, cpu state, storage content and network connections. The in-memory state and the content of CPU registers are copied. Then, this content is sent to the another hypervisor to be loaded into the VM. If source and target nodes are connected to a centralised storage, the storage content does not have to be migrated [97, 98]. With a decentralised storage, the migration process could overload the network, needing several minutes to move all the data from host to target node. Network connections that VM could have before the live migration do not present problems if VMs are not in different subnets. Otherwise, the applications could present disruption due to network changes.

The main requirement of VMs migration is the necessity of a Network Attached Storage (in short, NAS) uniformly accesible from all hosts. This technique does not allow VM images on local disks. The problems that a VM migration entails is the performance loss and the energy overhead. In order to improve those lacks, there is a wide scope of researching works in cloud computing environments. They can be classified into VM migration costs, processes and algorithms.

VM migration costs analyse the performance and energy consumed by the entire process of perform this action. In that context the total migration time and the downtime are the main tested parameters. The time passed from the start of migration process, until the virtual machine is resumed on destination host is the total migration time. It depends on the network bandwidth between host and target, and the total amount of memory that will be transmitted from source to target hypervisor. Total migration time is usually in a range from 10 to 120 seconds. Downtime is defined by C. Clark et al. as the last phase of VMs live migration process where the virtual machine is not running (stop-and-copy phase) [96]. Downtime is dependent on the workload of the application running in a VM. For specific workloads, downtimes can be of 60ms.

W. Liu et al. present a linear model to calculate the energy consumption of live migration [99]. They argue that if the amount of memory copied during migration increases, the energy increase linearly too. The model is validated based on five different benchmarks running on the migrated VM. The experiments show an estimation error below 10% and verify the effectiveness of the approach. However, Liu et al. did not recognise all parameters that may influence power consumption like the assumption of no co-located VMs on source as well as target host and therefore, no background load.

Huang et al. show that energy model increases non-linearly with the increment of the entire CPU utilisation of source and target host [100]. This work shows a new energy model for migration overhead which take the server entire CPU utilisation into account.

There are several works in VMs migration describing the improvement on existent migration techniques. Following research works show a reduction in downtime and migration time without decrease the performance of the applications.

W. Liu and T. Fang present a live migration approach based on checkpointing/recovery, trace/replay technology and an interactive mechanism with CPU scheduling [101]. They have built a model, implemented a prototype in Xen and achieved a performance evaluation in real applications. Also, they show a better average performance compared with pre-copy scheme (62.12% on downtime and 43.84% on total migration time).

J. Yang presents the framework *Magician optimisation migration*, based on the analysis of the memory transfer in real-time migration of current Xen virtual machine [102] . Yang adds a layered copy module and a memory compression module to the Xen VM. It optimises the time and space complexity to perform real-time transfers, improving migration performance of the VM.

L. Haikun et al. adopt checkpointing/recovery and trace/replay technology in *CR/TR-Motion* [103]. This technique provide a fast VM migration for LAN and WAN environments. With an execution trace logged in the source host, a synchronisation algorithm is performed to orchestrate the running source and target VMs, until they reach a consistent state. The migration overheads reduction is compared with memory-to-memory approach in a LAN: up to 72.4% on application observed downtime, up to 31.5% on total migration time, and up to 95.9% on the data to synchronise the VM state. The application performance overhead due to migration is kept within 8.54% on average. The results also show that for a variety of workloads migrated across WANs, the migration downtime is less than 300 ms.

VM migration algorithms analyse the best placement for VMs. The best placement depends on the criteria selected. Energy saving, the reduction of the carbon copy, or a better performance, are the most common parameters to optimise.

F.F. Moghaddam presents an intelligent live migration of VMs [9]. This work is focused in the calculation of the carbon footprint and energy consumption for the whole network and its components. Simulation results show that using the proposed Genetic Algorithm (GA)-based method for live VM migration can significantly reduce the carbon footprint of a cloud network, comparing it to an individual datacenter server consolidation. In addition, the WAN data center consolidation results show that an optimum solution for carbon reduction is not necessarily optimal for energy consumption, and vice-versa.

M.Cheng et al. propose a VM sizing approach called effective sizing [104]. It simplifies the problem associated to VMs dynamic load with a fixed demand. Effective sizing decides a VMs resource demand through statistical multiplexing principles. The authors of this work have designed a polynomial time VM placement algorithms for VM migration cost and cost-aware scenarios, which reach a 10% to 23% more energy savings.

S.K. Bose et al. propose strategies to optimise the placement of VMs, depending upon the cost of computation and the load at these locations [105]. They combine VM scheduling strategies with VM replication strategies. In particular, a selective replication of VM image across different cloud sites is proposed. Those replicas (primary copy), are updated with incremental changes. The replica placement strategies are based on factors that influence long-term costs such as the average per-unit cost of storage and the average per-unit cost of computation at different cloud sites besides the end-user latency requirements associated with the VMs. With the purpose of minimising migration latencies, associated with live migration of VMs across WAN, they design a replica placement algorithm that minimises additional storage requirements. The authors called it CloudSpider.

2.3 Modelling and simulating cloud computing architectures

In computer science, simulation is the technique for representing the real world by a computer program, which should imitate the internal processes and not merely the results of the thing being simulated. Robert E. Shannon defines simulation as *the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behaviour of the system and/or evaluating various strategies for the operation of the system* [106]. Simulation models can be classified along different dimensions.

A continuous simulation uses differential equations (either partial or ordinary), implemented numerically. These type of simulations are most appropriate if the material or information that is being simulated can be described as evolving or moving smoothly and continuously, rather than in infrequent discrete steps or packets. Discrete-event simulation concerns the modelling of a system as it evolves over time, by a representation in which the state variables change instantaneously at separate points in time. These points in time are the ones at which an event occurs.

If a simulation model does not contain any probabilistic component, it is called deterministic. In a deterministic simulation, a system is simulated under well determined conditions. In this kind of simulations, only one run is needed and there is no truly random variable involved. Furthermore, the output is determined once the set of input quantities and relationships in the model have been specified. Otherwise, stochastic simulation is used when random input values are needed. Stochastic simulation models produce output that is itself random, and must therefore be treated as only an estimate of the true characteristics of the model.

Simulations may be local or distributed. A local simulation is executed on a single computer. Distributed models run on a network of interconnected computers. Simulations dispersed across multiple computers are often referred to as "distributed simulations".

Generally it is difficult to ensure the real capabilities of cloud computing architectures, or an improvement over an existing one, until the environment is completely deployed. For this reason, it is very important to obtain an early accurate estimation of which will be the future performance of the new system. In order to achieve these goals, a widely used solution by the research community is to use models and simulation techniques.

2.3.1 Simulation tools for modelling distributed systems

During last years, simulation techniques are increasing their role becoming a widely using tool by the research community [107]. At present, simulation tools are essential for carrying out research experiments in distributed systems. These tools go from very specific component simulation to very large systems, like cloud computing systems.

For instance, in the networking scope we can find NS-2 [108], DaSSF [109], OMNET++ [110], and OPNET [111], among others. These simulators are focused on network details, such as network protocols, path discovery, latencies, or IP fragmentation, but lack the details to simulate virtualisation-enabled computing resources and applications.

Some examples of frameworks for simulating hosts completely are Simics [112], Gems [113], and SimFlex [114]. Simics is a full-system simulator developed by Swedish Institute of Computer Science (SICS). Simics is capable of simulate systems such as Alpha, x86-64, IA-64,

ARM, MIPS (32- and 64-bit), MSP430, PowerPC (32- and 64-bit), POWER, SPARC-V8 and V9, and x86 CPUs. Simics is also used as virtual platform for prototyping embedded hardware or new processors. Gems and SimFlex are simulators that depend on Simics. Gems is a simulation toolset to characterise and evaluate the performance of multiprocessors systems. SimFlex is a simulation framework which uses a component-based design and a rigorous statistical sampling, to enable the development of complex models. Also, this framework ensures representative measurement results with fast simulation times. Those simulators have detailed multiprocessor memory systems, but they lack of multiple-system capabilities.

SIMCAN is a simulation platform for modelling High Performance Computing (in short, HPC) architectures [115]. This platform is aimed to test both existent and new designs of HPC architectures and applications. SIMCAN has a modular design that eases the integration of the different systems on a single architecture. The design follows a hierarchical schema that includes simple modules, basic systems (computing, memory managing, I/O and networking), physical components (nodes, switches, etc) and aggregations of components using racks and blades.

In the past decade, Grids [116] grew up for give high-performance services to the scientific community. To support the researching scope, another set of simulators have been developed, such as GridSim [117], OptorSim [118], MicroGrid [119] and GangSim [120]. These tools can simulate brokerage of resources and the execution of different types of applications on different types of computing resources, but they also lack the details to simulate a cloud environment.

2.3.2 Cloud computing simulation frameworks

In cloud systems there are many factors that have a direct impact in the balance between the cost of energy/carbon footprint emissions and performance, such as the energy consumption of each component, the management of the virtual machines that are executed in the computing nodes and the algorithms for saving energy. However, predicting the impact of the energy consumption, even when small changes are applied in cloud systems, is a very difficult and non-trivial task. In general, the existing methods for measuring the energy consumption of individual hardware-component are non-scaling and intrusive, and in some cases, additional equipment is needed. Consequently, calculating the total energy consumption of executing a set of applications requires a lot of effort. Moreover, these methods quickly become impractical when scaling to large scale systems due to the cost of buying power measurement hardware and instrumenting each computing node. This results in trade-offs between cost, ease of management, performance, and power measurement detail.

The inherent complexity of cloud systems make simulation techniques of capital importance for acquiring knowledge about the elements that impact both the overall system performance and energy efficiency.

Due to the fact that cloud computing is a relatively new research area, there are few simulation frameworks that focus on modelling these systems. Some of them are Grid or HPC simulators that have evolved to simulate cloud computing capabilities - virtualisation and resources management. On the contrary, very few simulators have been created from scratch.

Next, some of the most well-known simulation frameworks are classified and described (see table 2.1). In order to perform this classification, nine relevant features for simulating cloud systems have been chosen to give a comparison of the simulation frameworks: *Language* represents the programming language used to program the simulation framework; *Open Source* means that the code is licensed to free redistribution and access; *GUI* indicates if the simulation framework have a Graphical User Interface (in short, GUI) for helping users to model and configure simulated environments; *Physical models* represent if the simulator provides mechanisms for modelling the hardware components of a node; *Communication model* refers to the communication components to perform simulations of distributed systems, such as data centers and cloud federation; *Virtualization model* indicates that the simulator has a virtualisation layer to define and simulate virtual machines; *User model* means that the simulator considers the possibility of multi-tenancy, i.e. simulating several users and giving the possibility of defining the resources and jobs for each user; *Scheduling model* indicates that the simulator provides methods for resource provisioning and user-job provisioning/scheduling on virtual machines; Finally, *energy model* refers to the possibility of performing simulations by measuring energy consumption.

To the best of our knowledge, the simulation frameworks that are best suited for the purpose of modelling and simulating cloud computing environments are CloudSim [121], MDCSim [122], GreenCloud [123], SimGrid [124], PVMsim [125], Virtual-GEMS [126] and SPECI-2 [127].

In the case of CloudSim, there are several research articles showing the obtained results on this simulator [128, 129, 130]. This tool was initially based on a grid simulator [121] (this being GridSim [131]). A new layer was implemented on top of GridSim to add the possibility of simulating cloud systems. Newer versions of CloudSim have been re-designed from scratch, and do not rely on GridSim anymore. A good feature is that CloudSim still has some drawbacks. For instance, there are no models for creating communications between virtual machines in several data centers. Another drawback is the fact that it is not possible to simulate overload in nodes. However, CloudSim provides CloudAnalyst [132], a GUI which allows the configuration of high level parameters.

MDCSim is an event-driven simulator similar to CloudSim. It has been designed as a pluggable three-level architecture. The simulator captures all the important design specifics of the underlying communication paradigm, kernel level scheduling artefacts, and the application level interactions among the tiers of a three-tier data center. MDCSim is capable of measuring power consumption across the servers of a multi-tier data center. However, this simulator provides mechanisms for modelling with a high level of detail disk drives, communication networks, schedulers and the application layer. On the contrary, MDCSim is not able to model with this level of detail components such as memories or CPUs. In addition, the main drawback when considering MDCSim is that it is not available for public download since it is built on CSIM, a commercial product [133].

GreenCloud is an extension for the NS2 network simulator [108]. GreenCloud is focused on simulating the communications between processes running in a cloud at packet level. In the same way as NS2, it is written in C++ and OTcl, being a disadvantage for this tool, since two different languages must be used to implement a single experiment. GreenCloud provides plugins that allow the use of physical layer traces that make experiments more detailed. For instance, a packet loss probability in the optical fibre depending on the transmission range, which can be obtained via simulation of signal propagation dynamics

and provided to GreenCloud.

SimGrid [124] is a simulator that provides core functionalities to simulate algorithms and distributed applications in distributed computing platforms, which go from workstations to grid environments . The resources are modelled by their latency and service rate. The topology is configurable by users. Collections of links may be used to simulate complex mechanisms like routers. The user is also responsible for scheduling computations and communications among the available resources. Simgrid is focused on modelling IaaS in cloud computing environments. Cloud simulations with SimGrid inherit the shortcomings of the previous versions of this simulator. Due to this simulator lacks hardware component details, it is not capable of providing simulations which include detailed access to hardware components such as main memories, cache memories. Low-level device driver policies can also have an impact on power consumption and performance on hardware devices. However, SimGrid is currently under development.

There are some simulation frameworks that are focused on the resilience of cloud computing, representing each node as a set of cloud services. The main issue of these simulators is the high abstraction level of the computing nodes. Basically, it hinders the possibility to perform low-level experiments and to obtain significant results. Some of these simulators are PVMsim, Virtual-GEMS, and SPECI-2.

PVMsim is a parallel multicore user-level simulator based on multi2sim [134]. The authors of this work present multi2sim as a simulation platform to study the behaviour of VMs over multiprocessor architectures. The main blocks of the simulation platform are the hardware virtualisation, the communication, and the synchronisation mechanisms. Each virtual machine is represented by a POSIX thread that can execute its own applications. This mechanism is achieved by launching a single functional simulator - m2s-fast - for each thread, in order to simulate a virtual machine. However multi2sim lacks mechanisms to model networks, focusing the platform on virtual machines management in single nodes.

Virtual-GEMS has been developed to simulate several virtual machines with multicore architectures [126]. It is based on Simics [112] and GEMS [135]. For this reason, Virtual-GEMS inherits modules written in C, C++, Python, and Ruby. Like PVMsim, Virtual-GEMS creates one Simics simulation process that represents a virtual machine per each GEMS instance - the memory timing simulation. On the one hand, virtual-GEMS has a complete structure to simulate checkpointing processes and virtual machine consolidation. On the other hand, this simulator does not include simulation networking capabilities and multiprocessor systems to support evaluation of complex cloud computing systems.

SPECI [136] - Simulation Program for Elastic Computing Infrastructure - is a simulator based on SimKit, which has been written in Java [137]. It is focused on exploring cloud-like scaling capabilities in data centers. SPECI consists of two packages. The first one is meant for datacenter layout, nodes, and network topology. The second one is meant to simulate components for experimenting with execution and measuring. SPECI is a light framework (15MB)) but it lacks hardware-level details for allowing user-customisation when performing experiments. The framework was updated in SPECI-2, where a data center is modelled in a hierarchical manner. It contains aisles, or clusters; each aisle contains racks; each rack contains chassises; each chassis contains blades; and each blade runs cloud services. SPECI-2 is focused on the resilience and the elastic properties of a cloud computing environment, with the same drawbacks as SPECI to customise experiments. However,

SPECI-2 does not provide a GUI to ease the parameters configuration of experiments, the post-processing after simulation experiments, and the graphics creation from the output files. It only provides a few scripts that support these functionalities.

		CloudSim	MDCSim	Green Cloud	SimGrid	PVMsim	Virtual Gems	SPECI2
Features	Language	Java	C++/ Java	C++/ OTcl	C/ Java	C	C/C++/ Ruby/..	Java
	Open Source	yes	no	yes	yes	yes	yes	yes
	GUI	yes	yes	yes	yes	yes	yes	no
Models	Hardware	limited	limited	limited	limited	full	yes	limited
	Communication	limited	limited	full	full	no	no	full
	Virtualization	full	no	no	limited	yes	yes	yes
	Users	limited	limited	limited	no	no	no	no
	Scheduling	full	no	limited	limited	limited	no	limited
	Energy	limited	limited	limited	no	no	no	no

Table 2.1: Summary of simulators models for cloud computing systems.

Table 2.1 summarises the comparison of the previously described simulators for cloud systems. Most of these simulators are focused on the behaviour of the networks in distributed environments, such as CloudSim, MDCSim, Green Cloud, and SimGrid. On the contrary, the rest of them are designed (or adapted) to simulate a single node behaviour and its virtualisation layer. Moreover, aspects like modelling the user's behaviour, resource management, and linking applications to resources to optimise performance are not taken into account. The notation for the terms of table are grouped depending on the completion of its implementation and models. If a simulator provides full implementation of given feature, the value is marked as *yes*; *limited* is used to indicate that this feature is not completely implemented; *no*, has been used for those features that are nor implemented neither modelled by the simulator.

From the simulators previously analysed, only CloudSim, MDCSim and GreenCloud offer mechanisms to perform experiments for measuring the electrical power. Moreover, the energy models provided by these simulators do not provide enough level of detail to model efficiently the energy consumption. Instead, they use different methods for measuring the energy consumption, like using consumption forms for CPUs or considering the node as an unitary consumption block. This is why the provided physical models are not suitable to perform accurate cloud system simulations, hindering in most cases the possibility to achieve energy experiments. In fact, the energy consumption of nodes is modelled linearly when number of active CPUs grows as well.

2.4 Summary

Cloud computing is a business opportunity for big companies in computer science and the chance for users shifting in a technology that offers elasticity, flexibility, and ad-hoc solutions. Furthermore, users may save money by avoiding obsolescence and maintenance of their own equipments.

As a result of a major usage of big data centers providing cloud computing solutions, the energy consumption is focused in a single point, the cloud providers. This ever increasing energy consumption of the data centers causes the problem of supplying enough energy for nodes, and also supplying enough energy for refrigerating each consuming node. Due to this fact, the researching area of energy aware and green computing have shifted, turning from increasing the performance of the data centers to minimise its energy consumption.

In this chapter we have shown the state of the art techniques and solutions for modelling energy consumption in computing systems, and the major simulation techniques and platforms existing currently for cloud computing systems.

One of the main issues for scientists and researchers is the complexity and cost of equipments and tools for analysing their techniques for saving energy. In addition, the measurement techniques may be invasive and costly. For this reason, it is necessary a simulation platform to model and simulate cloud environments and its underlying data center with the corresponding energy models to calculate the hardware consumption. Considering all the limitations exposed in existing tools for modelling and simulating cloud systems, a new platform should be designed to provide functionalities that implement those uncovered features.

Next chapter, *Modelling and simulating cloud computing environments* presents the iCanCloud simulation platform, a proposal of a flexible and scalable simulator that aims to study the trade-offs performance-energy consumption.

Chapter 3

Modelling and simulating cloud computing environments

This chapter presents our approach of a flexible and scalable simulation platform for modelling energy-aware cloud computing systems. The design of this simulation platform is aimed to build cloud models by configuring each part of the cloud system independently, that is, the hardware part of the cloud, management policies, virtualisation and energy consumption.

The new contributions presented in this chapter are propose a model for system architecture and propose a model for energy of system. On one hand, this design provides a cost-effective method to build a wide range of cloud configurations. On the other hand, large systems, containing a vast number of nodes, can be modelled in order to analyse the energy consumed by these nodes, and therefore, the total amount of energy required to sypply the complete system.

3.1 Introduction

Cloud computing systems are commonly supported by large data centers containing a vast number of nodes. Basically, those nodes provide two basic services: computing and storage. While computing nodes provide powerful CPUs to execute several VMs, storage nodes focusing on attending remote data accesses efficiently by using fast and high-capacity storage devices.

However, maintaining these systems require great amounts of energy. In this scope, a small reduction of the energy consumed by nodes may be reflected as great economic saving. Although data-sheets provided by manufacturers facilitate the calculation of the theoretical energy consumed by each node, the characterisation of the total energy consumed by the entire data center is a complex and difficult task. Also, the high and unpredictable number of users requesting resources concurrently hampers this task.

This problem is not linear and neither new for the scientific community. Several studies focusing on characterising the energy consumed in data centers are usually intrusive. In some cases, these methods involve directly instrumenting the motherboard with multimeters in order to obtain each power connector's voltage and current, thus obtaining real-time

power consumption [13]. The decision of performing internal modifications on nodes for saving money against the possibility of damaging, or ever loosing, the modified nodes, is difficult to take by the owners of a data center.

The raising complexity of cloud systems has made simulators a very helpful alternative for designing and analysing large and complex architectures. Dimensioning a data center, calculating its trade-offs or studying both actual and non-existent cloud computing architectures are some of the issues alleviated by using simulation tools.

Since the underlying architecture of cloud computing environments is transparent for users, main interest of these users rely on the pair costs-resources for executing applications and managing data. In the scope of cloud computing, where pay-as-you-go business model is the main link between user and the cloud, the quantity of rented resources and/or the time to complete executions may acquire great importance. Hence, simulators are specially useful for estimating the trade-offs between cost and performance, allowing users to estimate the best fitting of their requirements for a given budget.

In order to alleviate the previosly described drawbacks, a simulation platform, called iCanCloud, have been designed and modelled. The iCanCloud simulation platform is aimed to study and analyse the trade-offs between energy consumption and performance in cloud systems. The main advantages provided by iCanCloud are following summarised:

- *Scalability*: the size of the underlying cloud architecture can vary from several computers to thousand of machines grouped in racks.
- *Flexibility*: users can model and simulate a wide range of cloud computing configurations. Since the cloud system can be modelled by configuring each part of the cloud independently, its easy to combine pre-defined parts of the system in order to build different cloud models.
- *Non-existent systems*: users are able to model both actual and non-existent cloud architectures to estimate the trade-offs between performance and energy consumption.
- *Costless*: using the proposed simulation platform does not require specific hardware to be executed. Also, a cloud is not required for experiments.
- *Ease of use*: iCanCloud provides a GUI where the users can easily and quickly model complete cloud environments.

The simulated cloud systems are modelled by using a collection of modules representing the behaviour of actual components of real cloud systems. These modules focusing on modelling physical devices, like CPUs, disks, networks and memories, and software elements, like applications, VMs, file systems and scheduling policies. All these components are hierarchically organised within the repository of iCanCloud. Moreover, the scalable design of iCanCloud allows to include new models in this repository, widen the range of possible cloud systems built using iCanCloud. Thus, figure 3.1 shows the layered schema of the simulation framework.

The bottom layer of the figure consists of the hardware models layer. This layer basically contains the models that are in charge of modeling the hardware parts of a system, like disk drives, memory modules and CPU processors. Using those models, entire distributed

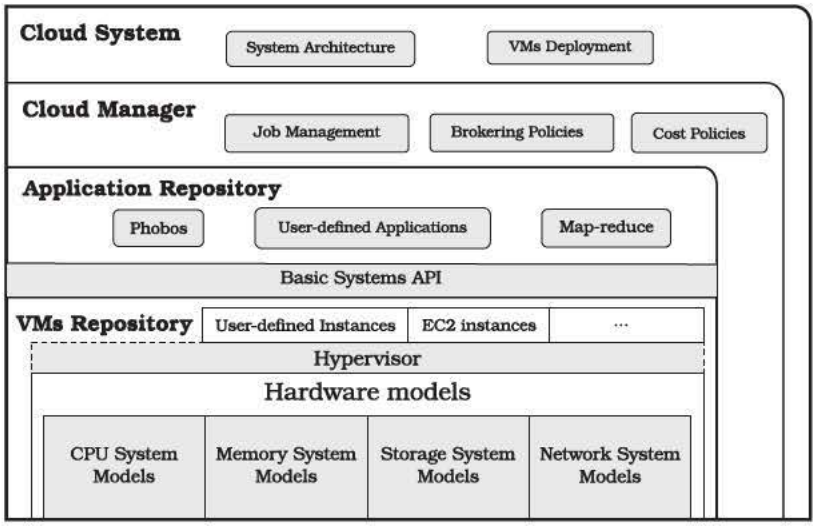


Figure 3.1: Basic layered schema of iCanCloud architecture

systems can be modeled and simulated. In turn, this layer consists of four groups, where each corresponds to a specific basic system: processing system (CPU), memory system, storage system, and network system.

The hypervisor is the layer on the top of the physical resources that is responsible for managing resources purchased by users. Therefore, it provides to users the abstraction layer that virtualise hardware, allowing to schedule VMs requests for accessing the devices, being the key of cloud computing systems. Thus, hypervisor scheduling policies that orchestrate accesses to hardware, may be created by users to analyse the impact of sharing the hardware system due to virtualization layer environments.

Upper layer consists of a VMs repository. This repository contains a collection of VMs defined by the user. It is possible to customize or add VMs models created by users defining non existent or existent VMs images, such Amazon EC2 instances, to the VMs simulation repository. Each VM is modeled by configuring the corresponding virtual resources purchased by users.

In a cloud system, the VM one of the main modules that defines a cloud computing system. Similarly, it is modeled as a building block for creating experiments by users. In these systems, as the schema exhibits, VMs are in charge of hiding the hardware details, providing to users a logic view that corresponds with the user requirements. Thus, the VMs models defined in this layer use the previously defined hardware components defined in the bottom layer.

Otherwise, the application repository contains a collection of pre-defined applications customized by users. Similarly to the repository of VMs, initially this repository provides a set of pre-defined application models. Those models will be used in order to configure the corresponding jobs that will be executed in a specific instance of a VM in the system. Moreover, new application models can be easily added to the system, because iCanCloud provides an API in order to ease the development of new application models.

The basic system's API module is directly connected with the hypevisor model layer. Basically this module contains a set of system calls which are offered as an API (Appli-

cation Programming Interface) for all applications executed in a VM model. Thus, these system calls provide the interface between applications and the services provided by hardware models managed by hypervisor. Moreover, it is possible to write applications to be simulated in the simulation platform using this API. In order to maintain a certain degree of compatibility, this API pretends to be a subset of POSIX. Moreover, a high level layer for developing distributed applications is designed. This layer is placed in the application component and provides standard interfaces for executing MPI applications.

Upper layer, called cloud manager, consists of a module in charge of managing incoming requests for purchasing resources as instances of VMs, where user's jobs will be executed. Once a job finishes its execution, the VMs where jobs have been executed are set as idle, and then re-assigned as available resources to execute the remaining jobs. This module may contain costs policies to assign VMs into specific servers by the corresponding heuristic.

Finally, at the top of the architecture is the cloud system. This module contains a definition of the entire cloud system, which basically consists on the definition of the manager, the definition of each VM that composes the system, and the definition of hardware that composes the underlying architecture of the data center.

However, all these components are hierarchically organised. Moreover, the model has been designed to be scalable, allowing to include new models in this repository, widen the range of possible cloud systems.

Moreover in order to show the architectural structure of the simulation platform at design level, Figure 3.2 shows main blocks and relationships which build up iCanCloud. It has been designed using submodels that contains a essential part of a Cloud Computing environment. Each submodel that represents a basic subsystem, that is, computing, memory, storage, PSU (Power Supply Unit), and network, is connected to an Energy Meter module. Basically, this module calculates the energy consumed by the hardware device linked to it. In turn, each Energy Meter is linked to the Energy Manager, which is in charge of gathering the amount of energy consumed by every component in the cloud system. Therefore, the Energy Manager manages the energy consumed by individual hardware components, the aggregated consumption of each computing node and the overall energy consumption in the entire cloud system.

Each node contains a Hypervisor module. The underlying design of the Hypervisor module achieves resource virtualisation and multi-tenancy, where jobs of different tenants that are running on the same computing node can share resources, while their jobs are isolated from each other.

The Cloud Manager is the master key of the simulation core. This module is directly linked with the Energy Manager and with each VM in the cloud. The main objective of the Cloud Manager is to allocate VM's, requested by tenants, in the available computing nodes in the cloud. Also, due to Cloud Manager contains information about the energy consumption of each node, it can use different intelligent scheduling algorithms in order to optimise trade-offs between energy consumption and performance. In fact, the design of this module allows users to interchange different scheduling policies, or even write new ones, to analyse both performance and energy consumption of the cloud system.

From now on, the term *user* is used to refer the person who uses the simulation platform, and *tenant* refers to the person who purchases services of the modelled cloud

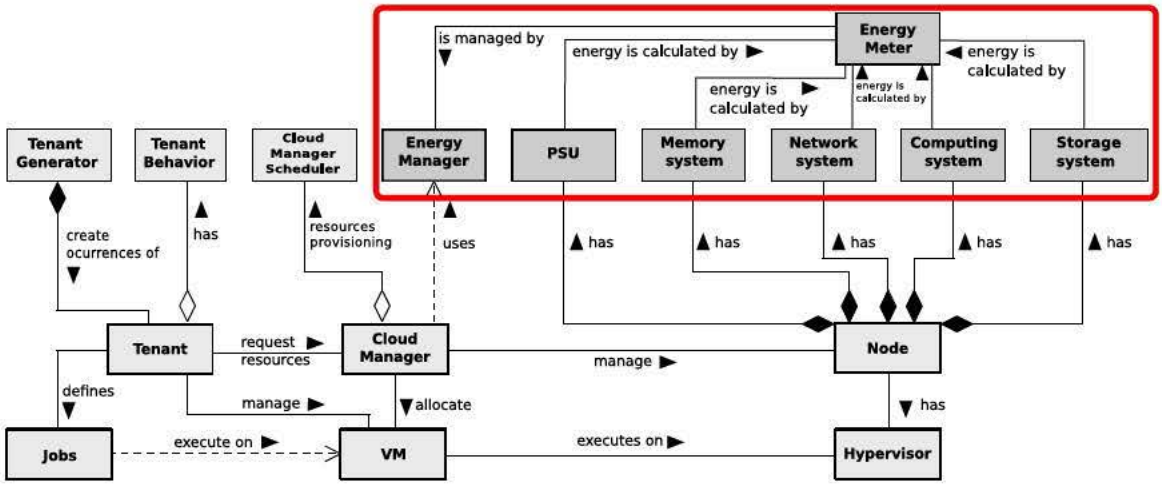


Figure 3.2: Class diagram representing the main architecture of iCanCloud

in a simulated environment. Each tenant in the system is individually modelled by using the Tenant module, where a tenant is defined by its behaviour and the set of jobs that are executed in VM's. Basically, this behaviour represents how each tenant executes jobs in VM's. For example, we define *FIFO tenant* as a tenant that uses a FIFO policy to map their applications to VMs, such that the first VM purchased is used to execute the first job defined by this tenant. On the contrary, we can also define *intelligent tenant* as a tenant that selects the most powerful purchased VM to execute the job that requires more resources. Finally, the Tenant Generator module is aimed to create tenants following an statistical distribution to simulate their arrival in the cloud system.

In order to provide a high level of flexibility, those models representing actual cloud systems, built using iCanCloud, have been structured in four independent sections: physical resources, energy consumption, virtualisation of resources, and management of the cloud system.

3.2 Modelling the basic subsystems of a node

Cloud systems consist of several components, both hardware and software, such as CPUs, memories, communication networks, hypervisors, VMs and resource provisioning policies. These components are combined to build nodes, it being those nodes also combined to build data centers. In order to properly simulate realistic cloud systems, the models that simulate these components are combined to build nodes, using the same schema as the real cloud systems.

These nodes are connected through a communication network, allowing users to configure the topology of each modelled cloud. Generally, large systems group nodes in blades, and these are grouped in racks. Hence, different cloud configurations can be easily modelled, like private scientific clouds and large storage data centers.

Figure 3.3 shows the section of iCanCloud focusing on modelling the hardware part of cloud systems, which is highlighted with dark grey boxes. Those models, that represent the main hardware devices of a node, are CPU, Memory, Storage, Network, Power Supply

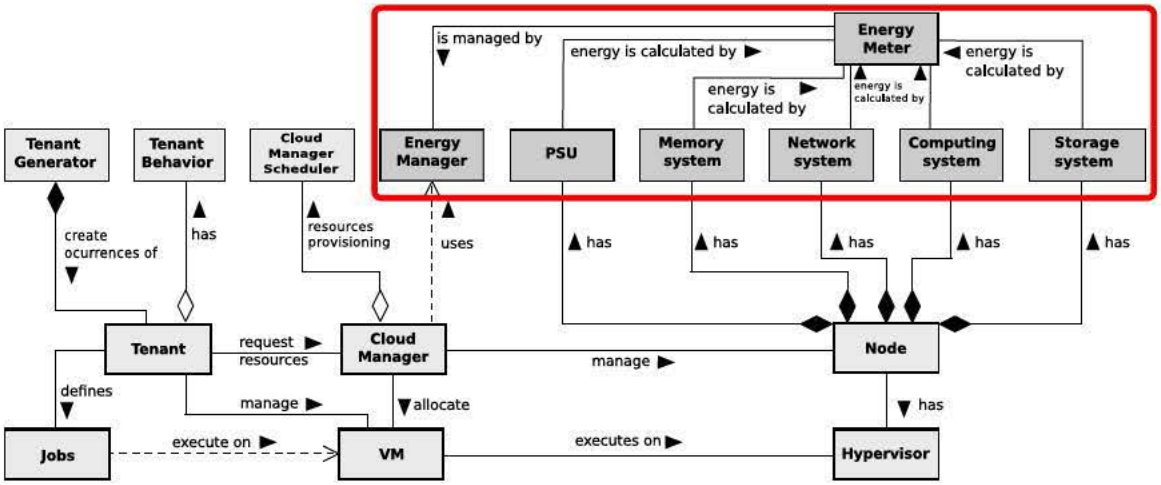


Figure 3.3: Class diagram focusing on the design of a node in iCanCloud

Unit (in short, PSU) and the Energy Meter.

The physical view of a node containing the basic systems, modelled using iCanCloud, is depicted in Figure 3.4. In order to process the requests sent by the applications, the Operating System (in short, OS) has been modelled as a module that contains the software components for managing each basic systems. The OS API is the center module of the OS, focusing on redirecting each request to the corresponding service. All physical subsystems are linked to the PSU, which is responsible for managing their main energy states depending on the global state of the node. For example, if the node is in *off* state and it is required to switch it to *on*, the PSU will send a signal to each device in order to change its state to *active*.

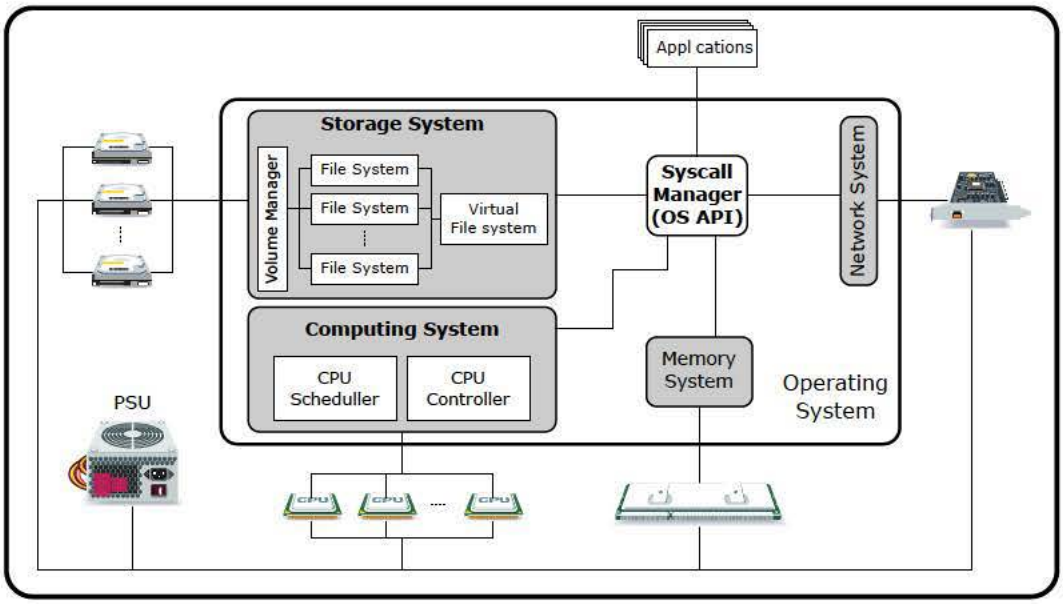


Figure 3.4: Model of the internal subsystems in a node

Since iCanCloud allows to fully customise the configuration of each node, heterogeneous cloud architectures can be modelled in order to represent the behaviour of realistic cloud systems. Figure 3.5 shows an example of an heterogeneous data center. This data center consists of three different types of nodes. The right part of the schema shows each node type and its internal parts. Usually, This heterogeneity of a data center is caused due to several reasons, like the obsolescence of equipments or a dimensioning of a data center. In both cases more computational resources must be added to the system. These new nodes are targeted to provide a basic service, such as data storage, where computational resources are minor but storage resources needed are higher, or on the contrary, attempting more computational resources where storage are less important against a powerful computing system.

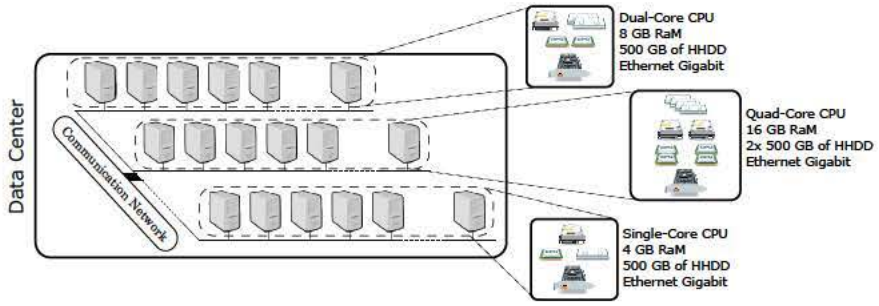


Figure 3.5: Example of an heterogeneous cloud system modelled using iCanCloud

3.2.1 CPU Model

The model that represents the CPU system in iCanCloud consists of three parts: a scheduler, a set of cores and a processor controller. The main objective of this model is to calculate the time required to execute a specific number of instructions. In iCanCloud, these instructions are grouped in *computing blocks*, which are measured in millions of instructions (in short, MIs). Thus, different applications may request to execute different computing blocks in a shared CPU.

The CPU scheduler manages those computing blocks that arrive to the CPU system, in order to execute them in a CPU core. Thus, the main objective of this module is to select, following a pre-defined strategy, both the computing block and the CPU core where this computing block is going to be executed. Hence, the strategy used by the scheduler may allow parallelism between applications, for example, executing different blocks in different CPU cores, or interleaving computing blocks in a single core.

iCanCloud provides three different scheduling strategies for managing computing blocks: FIFO, Round-Robin, and priority-based. Moreover, new strategies can be added to the simulation platform [138, 139].

Each CPU core calculates the time required to execute computing blocks. The portion of each computing block to be executed is set by the CPU scheduler. The computing power, measured in millions of instructions, is calculated for each CPU core by the following formula:

(3.1)

where MIPS are the millions of instructions per second that the core is able to execute and is the frequency of the core. This frequency defines the time interval between ticks, which is the maximum amount of time that a block needs to be processed by a core without being interrupted.

However, a CPU core can change its execution state. Depending of the load of a given core, the effects of those states may vary: switching on/off the node entails the same effect in the state of each core; C-States attempt to obtain consumption fluctuations turning on/off parts of the processor, by moving the core states between different power saving modes; P-States modify the frequency of processing in order to obtain different performances and variations of energy consumed. These states fluctuations have been modelled within the CPU Controller, which manages on demand, the states and the transitions between them for each core.

The CPU Controller uses the concept of dependency for managing these states. In a CPU, dependency is defined as the cohesion grade between one core and the rest of cores that composes the processor. Therefore, if a core changes its state, this change may affect to the rest of the cores (dependent), it does not affects (independent), and only some changes affects to all the cores (partial). The dependency between the states of the core can be customised by users.

Besides the dependency between cores, the behaviour of P-States has been modelled to change the of the core that is being modified. For example, a reduction increments the cycle of ticks per second and, consequently, less IPS are executed by each core. This technique is applied in low workload states, producing a lower energy consumption. On the contrary, increasing the frequency causes a fast processing of computing blocks.

The design of the CPU system allows user to create new CPU models by defining a set of states, the number of performed ticks per second and the power required to set the CPU in each state. Thus, this structure allows to model both existent architectures of CPUs, such as dual-core, quad-core, or hexa-core, and non-existent ones.

In order to illustrate the behaviour of each modelled system, different diagrams have been provided. For the sake of simplicity, the same notation have been used in these diagrams. This notation is described as follows: circles show an entry point of sub-state machines, where the circle placed at the top represents the beginning of the diagram; the concentric circle is the finalisation of the diagram; unitary squares show actions; squares containing an initial point show different use cases; arrows show the transitions between different actions or transitions between an action and another module; diamonds represent conditions; finally, thick lines represent parallel actions.

Figure 3.6 describes the behaviour of the computing system used in the iCanCloud simulation platform. This diagram is divided in three main parts: a CPU Scheduler (left), a CPU Controller (bottom-right) and CPU cores (top-right). The module placed at the top of the CPU scheduler contains the scheduling policy. Basically, this module process the events handled by the scheduler and then it calculates the next computing block to be executed.

It is important to remark that CPU Core and CPU Controller contains different sub-state machines that are only activated by those decisions taken by CPU Scheduler.

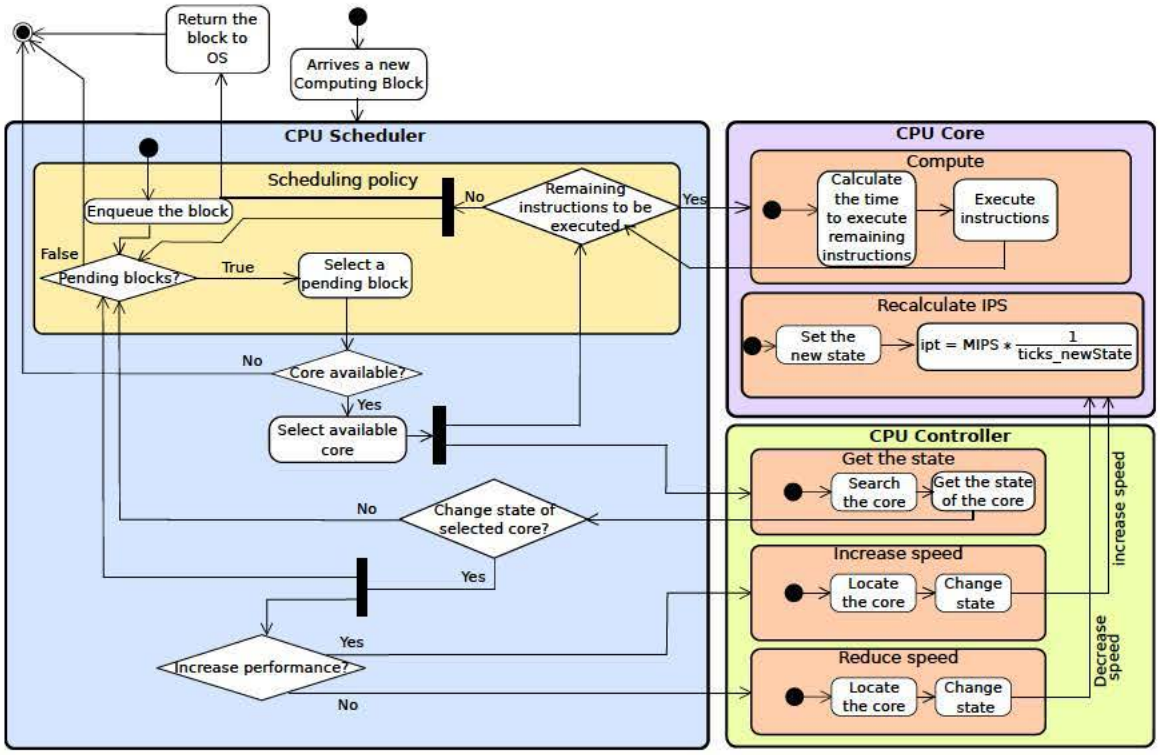


Figure 3.6: Modelling of the computing system

The sub-state machine allocated within CPU Core, called Recalculate IPS, shows how the performance states of each core are modulated by modifying their frequencies. The equation used to perform this task is defined as:

$$IPT = MIPS \times \frac{1}{ticks\_newState} \tag{3.2}$$

where IPT is the new value (Instructions Per Tick) used in the new state of the CPU core, and $ticks\_newState$ is the number of ticks per second performed by the core in its new performance state.

Figure 3.7 shows an example of how our proposed simulation platform simulates the computing system using a FIFO algorithm. Basically, the computing system is characterised by using three components, a CPU scheduler (left), a CPU controller (right) and CPU cores (bottom). In order to represent the state of each computing block we use two different colours, while black represents the part of the computing block that has been executed, white represents the part of the building block that has not been executed yet. Those computing blocks allocated in the scheduler are waiting to be executed in any of the CPU cores. On the contrary, those computing blocks allocated in a CPU core are being executed.

Figure 3.7.a shows a snapshot of the computing system where both $Core_1$ and $Core_3$ are executing a computing block. The CPU scheduler contains two computing blocks, B_1 and B_2. The CPU controller manages the state of each CPU core. At this point, $Core_1$ and $Core_3$ are in A *state*, while $Core_2$ and $Core_4$ are off. The CPU scheduler ask for

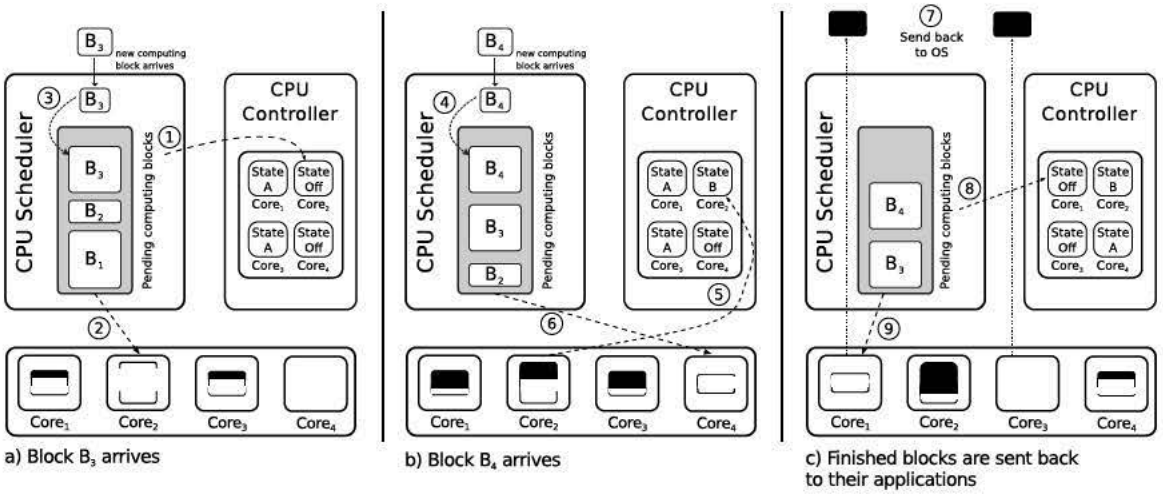


Figure 3.7: Example of the CPU system modelled using iCanCloud

an available core to the CPU controller ①. Hence, $Core_2$ is assigned to execute the next computing block, that is, B_1 ②. Finally, a new computing block arrives to the computing system, it being this block B_3 ③.

Next, a new computing block arrives to the computing system, B_4 ④ (see Figure 3.7.b). The CPU Controller changes the frequency (f_{clk}) of $Core_2$, setting its state to *B state*, in order to increase the number of instructions executed per tick ⑤. Both $Core_1$ and $Core_3$ continue their execution in *A state*. The CPU scheduler assigns $Core_4$ to execute the next computing block, that is, B_2 ⑥.

Figure 3.7.c shows the last snapshot of this example. At this point, both $Core_1$ and $Core_3$ have finished to execute their computing blocks. Therefore, these blocks are sent back to the operating System ⑦. The CPU scheduler checks for an idle core ⑧ and then it assigns $Core_1$ to execute B_3 ⑨.

3.2.2 Memory model

The memory system has been modelled using two different modules: a memory module and a memory manager. The memory module represents the physical characteristics of the main memory, it being responsible for simulating features like the size of the memory, bandwidth and latency times. The memory manager is in charge of modelling the structure of the memory, that is, the blocks that contains data. This module is also in charge of managing requests for allocating, and freeing, memory pages. Moreover, the memory manager could deny a request if the quantity of free memory is minor than the memory requested by the application. On the contrary, when the manager receives a request for freeing memory, it updates the number of free blocks in the system.

Basically, the main memory has modelled as a finite set of contiguous blocks. Using this logical view, each memory block represents a memory page. The size of the memory page and the total number of pages define the total size of the modelled memory. Moreover, different latencies have been also considered for modelling the main memory, such as reading, writing and searching for a memory page.

Using this model, the analysis of the scaling up of applications by calculating the amount of memory requested by each application is possible. Figure 3.8 shows the modelling of the memory system used in iCanCloud.

3.2.3 Storage model

The iCanCloud simulation platform provides a customisable storage system that consists of different modules (see Figure 3.9): a virtual file system (in short, VFS), file systems, a volume manager and storage devices. Each one of these modules can be customised individually. Therefore, iCanCloud allows to create a wide range of storage configurations, including parallel file systems, remote storage partitions and RAID systems. The basic idea for modelling the storage system lies in the translation of the data requested by applications to the list of blocks that contain such data. The complete behaviour of the storage system modelled in iCanCloud is described in Figure 3.10.

Virtual file system

The provided model of a virtual file system allows tenants to use different file systems. Basically, the main objective of this module is to filter data requests from different tenants

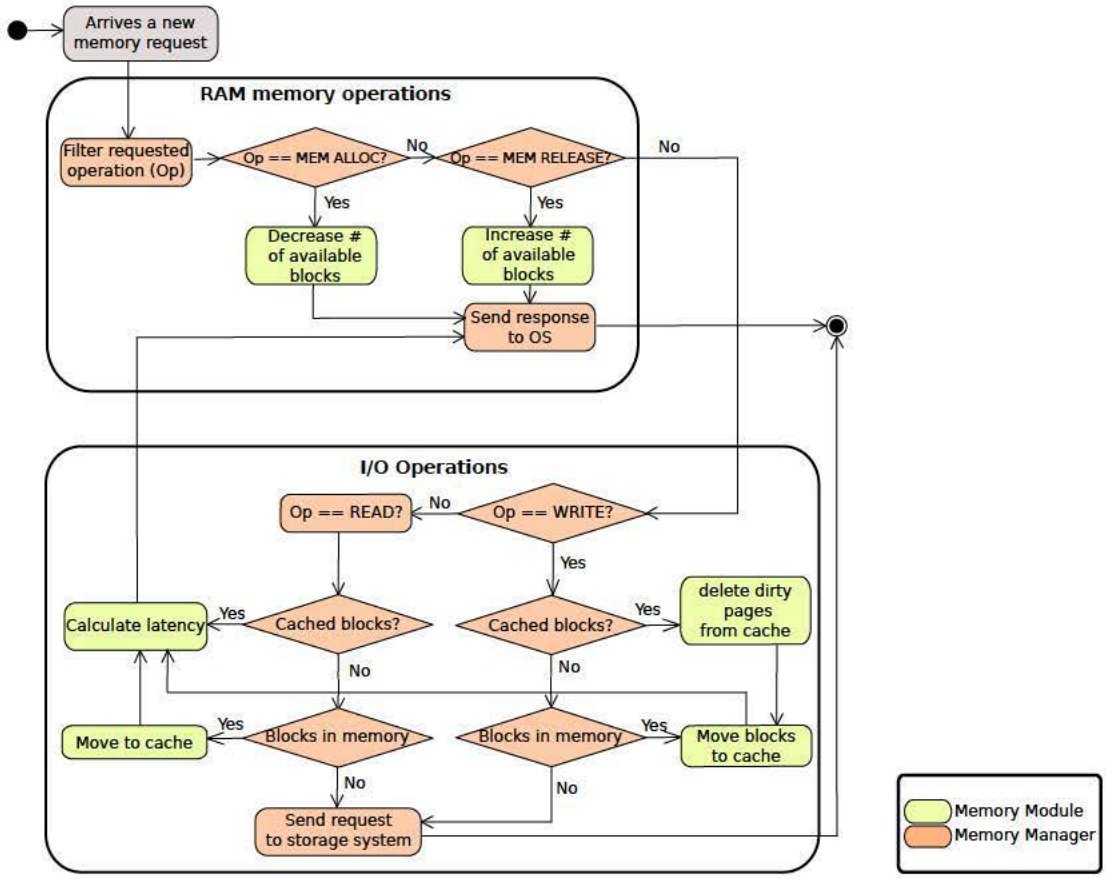


Figure 3.8: Modelling of the memory system

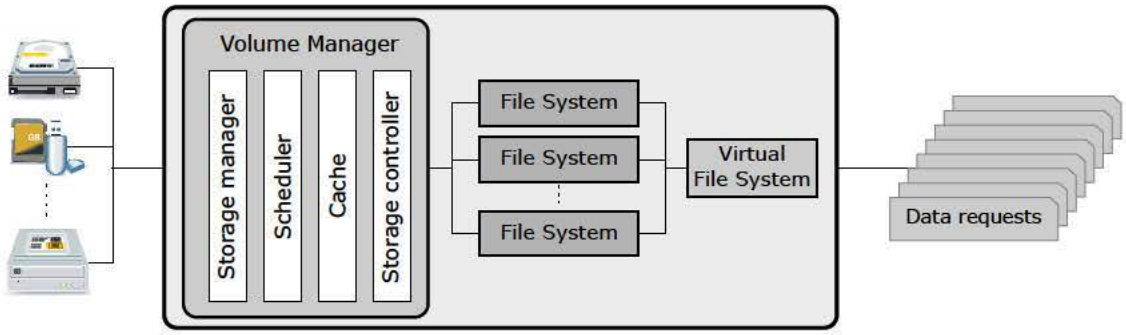


Figure 3.9: Basic schema of the storage system

and redirect each request to the corresponding file system. Since each file system is mounted in a specific path in a shared directory tree, this task can be easily achieved by analysing the absolute path generated by the incoming request.

Since the storage system would be used in a virtualised environment, it must allow tenants to create and destroy file systems each time a VMs is deployed in a node to be executed or, by the contrary, when VMs are shutting down. This design provides a high level of flexibility to create customised storage configurations, allowing to mount different file systems in remote storage servers, in run-time. The virtualisation part of this system is explained in more detail in Chapter 4.

File system

A file may be defined as the set of data blocks that composes of unitary logical structure created by user to storage data. Accordingly to this definition, the file system is the part of the storage system responsible for translating data requests into a list of blocks containing such data.

Since iCanCloud has been designed towards providing a high level of flexibility, this storage system allows the development of different modules that implements the functionality of a file system. Thus, different strategies can be implemented to perform this translation. Similarly, in other simulation platforms, different models to simulate file systems have been developed in the past [140].

However, in the current version of the simulator, a basic strategy has been implemented to represent the behaviour of file systems. Basically, this strategy consists in calculating the amount of blocks required to satisfy a given data request. The main advantage of this strategy is performance. Instead of using a list of those physical blocks containing the requested data, only the number of blocks are required, which avoid a lot of computation for each data request. Consequently, the level of fidelity provided by using this strategy may be slightly decreased.

The main reason to use this strategy is the complexity of the modelled systems. Since cloud systems may consist of thousand of nodes, using a list of blocks per data request in each file system may slow down the simulation considerably. Also, since iCanCloud provides a detailed layer of virtualisation, implementing and debugging highly detailed file systems may require a considerable effort in both debugging and testing phases.

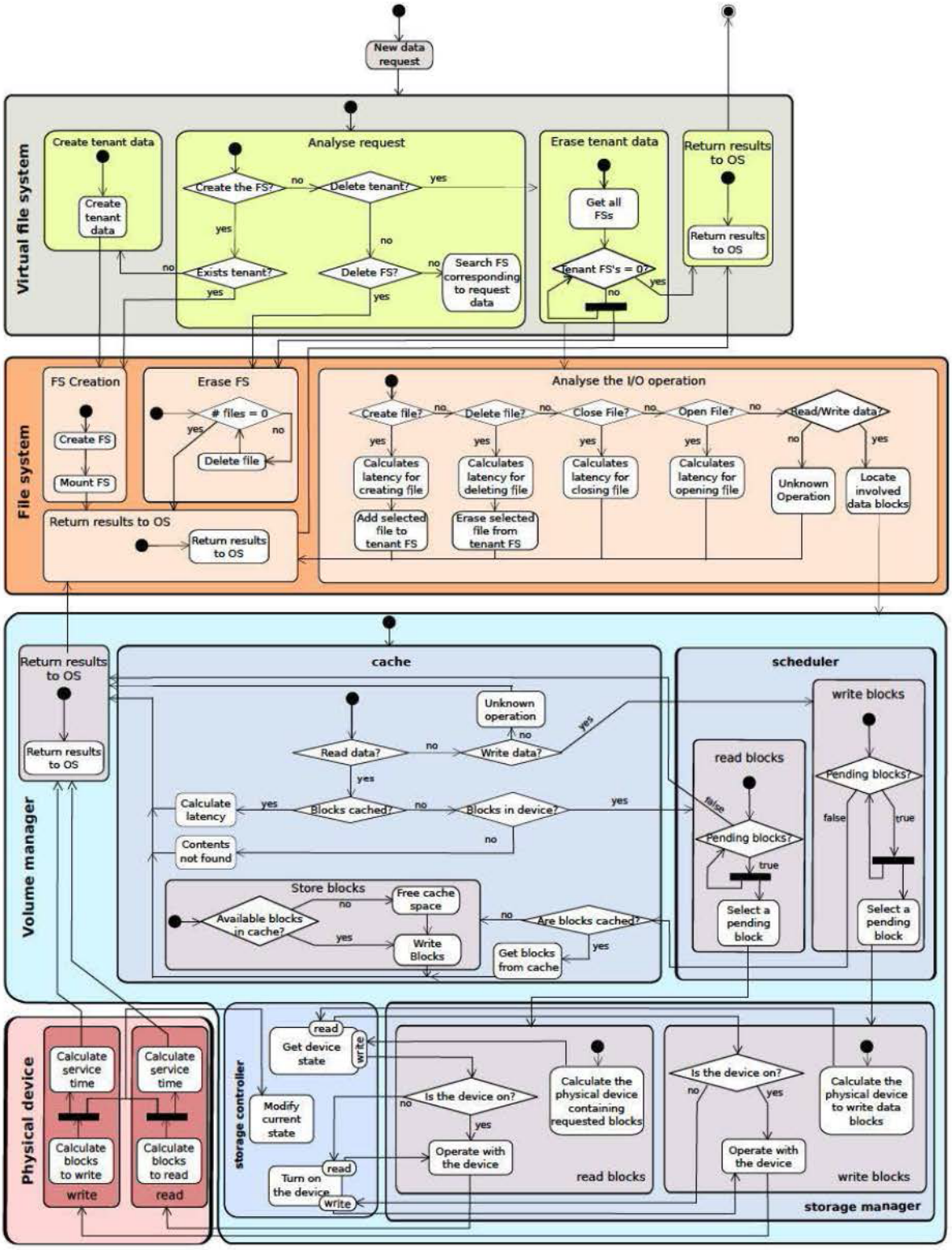


Figure 3.10: Modelling of the storage system

Volume manager

The Volume manager is the part of the storage model responsible for managing read and write operations of data blocks. Basically, this component receives data block requests from file systems and locates the data to perform a read or write operation. In iCanCloud, the volume manager has been modelled using four modules: a cache memory, a scheduler, a storage manager and a storage controller.

The model of the cache memory consists on maintaining the most frequently used data blocks managed by the volume manager. Then, those blocks requested from the file system that are stored in this cache, will be processed in this module, performing faster the incoming requests from file system. This models implements an statistical strategy to calculate if current processed block cached using a customisable hit ratio.

The scheduler manages requests of those data blocks that are not stored in cache. Depending on the modelled architecture, this policy may be customised by the user to build the behaviour of strategies such as FIFO, elevator algorithm, CSCAN and new strategies for managing data blocks.

The block manager is in charge of redirecting each data request to the device containing such data. This module can be configured for using several storage devices. For example, if the storage system contains several hard disks, this module may operates using those disks as a single volume, like a RAID system.

Finally, the Storage Controller is responsible for controlling the states of the physical devices. Similarly as CPU controller manages the state of CPU cores, Storage Controller manages the states of storage devices. Moreover, the user may customise the transitions between states, adding or removing states, in order to configure a specific storage device.

Physical devices

Finally, the physical part of the storage system consists of physical devices. The main task of these modules is to calculate the time required to operate with the set of the requested data blocks. Basically, these modules performs two different operations, read and write. Thus, once a request arrives, the device analyses the operation. While read operations demands returning the read block, write operations only return the result of performing this operation.

The iCanCloud physical device model allows to simulate any storage device by defining the corresponding set of parameters that represents its physical features. Those parameters, that are usually provided by manufacturers, have to be included in the simulator by users in order to model different physical drives.

3.2.4 Network interface model

The network is paramount to a cloud computing environment. The INET framework [141] has been leveraged in order to simulate a network system accurately. It provides a complete set of modules for simulating a complete network stack, architecture models, and configurations to simulate wired networks. However, its main drawback is the complicated configuration and use. In order to ease the use of network services, we provide a POSIX-like API. This API is used by tenants to manage their virtual machines and applications. The

cloud manager uses this API as well when manage resources, such as remote data storage.

3.3 Modeling the energy system of a node

At present, the energy consumption and its management in cloud computing environments is one of the main challenges faced by the research community. In order to model the energy consumption, several alternatives have been explored to adequate this charastecristic to the simulation platform.

An easy energy consumption model consists in including a global consumption estimation of the nodes depending on their states: idle, off, and running applications. Although this method is easy to model and provides a good performance, the results provided contain a significant error. This is mainly caused because this model only consider a uniform workload, which does not represent real cloud systems. Other methods consist in embedding average consumption values of hardware components to perform experiments. This method, however, does not take into account varying resource utilisation because it only works with constant averages. As shown by Barroso and Holzle [58], resource utilisation greatly influences the energy efficiency and energy proportionality of computing systems. In addition, neither of the described methods are flexible and scalable to allow user-customisation of simulation experiments by changing how power consumption values are calculated.

The model of iCanCloud, aimed to improve these gaps, is based on calculating power consumption values by modelling each component separately, in order to obtain the total power consumption of each hardware device in the cloud system. This, of course, not only includes computing systems, but also networking equipment, it being all of them simulated as well. Thus, each model that simulates a hardware device in the cloud consists of two modules. The first module, called component model, simulates the underlying behaviour of the modelled device. The other module, called energy meter, calculates the amount of energy consumed by the modelled device. In turn, the energy meter consists of two different modules: energy states and a consumption calculator (see Figure 3.11).

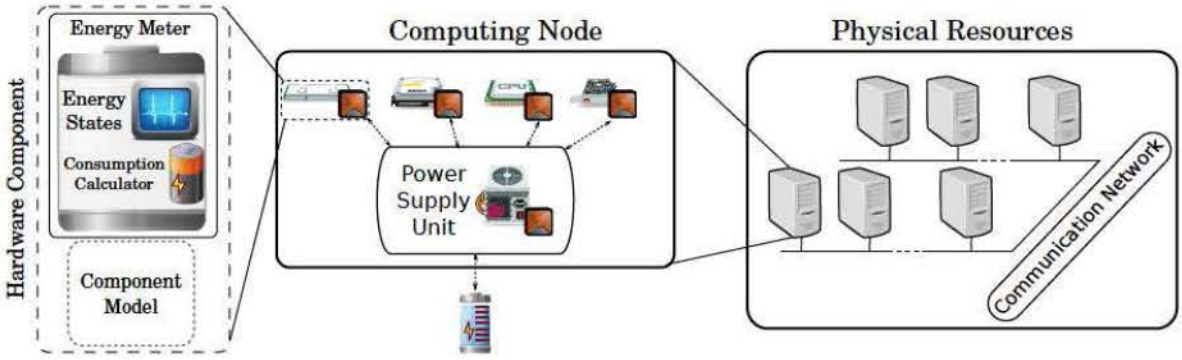


Figure 3.11: Energy consumption mechanish

3.3.1 Energy states

Every real hardware device such as memories, hard disk drives, network interface cards (in short ,NIC), power supply units (in short, PSU), and CPUs, have a set of states. Some of

the basic states in the case of the memory are pre-charge, activate, read, write and refresh. The hard disk drives states are writing, reading, seeking, starting, transferring, idle and off; the PSU and NIC ports have on/off states. In the case of the CPU, states can be grouped in two subsets: C-states, which save energy by cutting the clock signal from idle units inside the CPU, and P-states where the clock frequency is modified. The latter also affects the performance of the CPU when executing computational blocks. In a execution over a real system, a hardware device can use one state, several states, or none. In order to simulate realistic cloud systems, the hardware models in iCanCloud use different states, like real hardware devices. Hence, iCanCloud calculates the energy consumption in every hardware device managing and processing these states. In order to perform this task, three parameters must be configured: the states of the component, the time that the component resides in each state, and the power consumption of every state.

3.3.2 Energy meter

The energy meter module is responsible for loading, managing, and calculating consumption values and states of hardware devices. Before a simulation is started, the user has the option to activate the energy meter to enable energy consumption capabilities of each component. If this option is activated, the cloud manager module creates an energy meter module per component during the initialisation phase. Once the energy meter is initialised, it loads the consumption states and values that were configured by the user. Hence, once a simulation is performed, the time that the component stays in each state is recorded.

Moreover, the energy meter has been designed to allow users modelling their own hardware devices. Thus, entire data centers can be modelled by using both actual and non-existent devices. In order to model existent devices, data sheets provided by the manufacturer of these devices are needed to configure both performance and energy consumption values in the simulated model.

Next, we describe the models for calculating the energy consumed by each systen.

CPU energy model

The model for obtaining the CPU power consumption is based on L. Minas and B. Ellison [39]. In this model, the power consumed by each state of the CPU is calculated according to the following formula:

_ —— (3.3)

Hence, the estimation of total consumed energy ( ) given a processor utilization rate ( ), is computed using the maximum power consumption value ( ) and the idle state ( ), according to the formula 3.4. The energy consumed by the CPU for a given state is obtained by calculating the integral of the power consumed by the CPU, using as integration limits the first instant of time (recorded by the energy meter), and the total amount of time that the processor remained in that state ( ).

_ _ (3.4)

The total amount of energy consumed by the processor in each of the S states is modelled with the following equation:

(3.5)

Memory energy model

The most feasible method for estimating SDRAM power consumption is to use the current, voltage, and timing constraint values from the SDRAM data sheets, which are based on real measurements. The relationship between SDRAM utilization and power consumption is highly detailed. It consists of 20 formulas to model the states and transitions between them [45]. In order to obtain the memory power consumption, the model is based on the data sheets specifications for the DDR standard, which has a current estimation used to perform each state [46, 47]. The iCanCloud estimation of energy consumed by memory is modelled as:

(3.6)

The energy consumed by memory devices ( ) is calculated using the SDRAM memory states, which are pre-charge (PRE), activate (ACT), read (RD), write (WR), and refresh (REF). The power consumption for each state is obtained with Ohm's law . Data are obtained from the instant the memory is activated ( ) to the moment the simulation ends, time at which the total accumulated time of each state has been recorded (, , , and).

Network energy model

The power consumption of a NIC has negligible variability. It mostly depends on the number of active and inactive ports, as described in the 802.3az standard, and the network bandwidth rate [142]. Thus, power consumption of a NIC is calculated by:

(3.7)

The estimation of energy consumed ( ) for a NIC consisting of interfaces, is calculated as the power consumption during the time it remains activated , from to .

Disk energy model

Due to the growing amount of data generated by scientists and researchers, saving energy in disk drives is a hot topic. There are several alternatives to modelling hard disk drives [50, 39, 51]. The model used by iCanCloud to calculate an estimation of the energy consumed

by the hard disk drives is based on the methodology proposed by A. Hylick et al [52]. They detail the relationship of hard disk utilization and total power consumption by a server using performance data. Hylick relates that the trade estimator of the energy consumed by a hard disk drive can be obtained as a composition of active energy (read/write operations), seek energy and idle energy. The seek energy is less than 0.2 J. Therefore, it has been discarded for the energy model of iCanCloud. The energy during active state is obtained from the following equation:

— (3.8)

where is energy in Joules; *S is the file size in MB;* B is the bandwidth in MB/s, and is the active power, in watts provided by the manufacturer. is the time during which iCanCloud considers that the drive remained in active state. The energy consumed when the device is in the idle state is calculated from:

(3.9)

where is published by the drive manufacturer in the datasheet, and represents the time that the disk is neither active or powered off.

The trade estimator of the total energy consumed by a drive servicing a set *( )* of *N* requests, including *I* (time idle) is obtained from (3.8) and (3.9). It is calculated by:

(3.10)

3.4 The energy loss. Modeling the power supply unit

The PSU represents the third part of power consumption by a node. It is responsible for transforming alternate current to direct current. During this procedure, a significant fraction of power is lost. For this reason, most commercial PSUs are classified by energy efficiency [49]. This classification is used to calculate the consumption model of PSUs. In order to obtain a power consumption model, the power consumption of the hardware components, the rated output power of the PSU, and the percentages of efficiency supplied by PSU, are required. Thus, PSUs are modelled using the following formula:

——— ——— ——— ——— (3.11)

where the power consumption used by hardware components ( ) is obtained from (3.5), from (3.6), from (3.7), and from (3.10). This operation is required due to the fact that the obtained energy is in Jules, while should be the average power in Watts. Percentage of load ( ) is calculated using the following formula:

——————— (3.12)

– –

iCanCloud calculates the energy consumed by PSU ( ) using its rated output power, which is detailed in the manufacturer's datasheet. The datasheet shows the energy efficiency ( ) at different peak load utilization rates (20%, 50% and 80%), and the real load that hardware produces ( ). Therefore, the ranges selected to obtain the energy efficiency percentages are:

$$\begin{aligned} &\text{if} && , \\ &\text{if} && , \\ &\text{if} && . \end{aligned} \tag{3.13}$$

Finally, the energy consumed by PSU ( ) is modelled using (3.11) and (3.13). The result is:

$$\text{———————} \tag{3.14}$$

The iCanCloud consumption model for each component is fully customizable by the user. It is possible to modify, change or create new models by increasing complexity (or simplifying) depending on the desired accuracy of the experiments.

3.5 Energy manager model

It is needed mechanisms to collect the energy consumed per node. Energy meters linked to each component provide the consumption based on accumulated values. The energy manager is responsible for managing the underlying arquitectural system power, collecting the consumption values of each device that composes of a node. This module has two fold: it is in charge of obtaining the instant power that an energy meter is measuring, and it is capable of calculate the total power consumption of a node using historical data gathered from each energy meter.

In order to calculate the aggregated power consumption of a node, the energy manager collects the consumption values from each hardware component. The consumption models for the hardware components are described at Section 3.3.2. Using from (3.5), from (3.6), from (3.7), from (3.10), and from (3.14), the model for the global, aggregated power consumption of a node ( ) is obtained. The aggregated power consumption of a computing node is modelled using the following formula:

$$\tag{3.15}$$

where represents the energy consumed by the motherboard and the node's internal cooling system, being calculations obtained by a power meter or by a mathematical models.

We define node state as a transition that, when it is produced, all internal hardware components change their state to another one. For example, if a node is in off state, when the cloud manager starts it up, the CPU, memory, disk, NIC and PSU change their states from off to the equivalent state that each component has, such as: idle or active, depending on the component. The states manager, as part of the operative system of the node is responsible for performing this task, notifying state changes to all hardware components.

The energy manager offers an API to be used to perform different resources provisioning policies depending on consumption values, QoS, or/and calculate performance values of each node.

3.6 Summary

This chapter presents the models for simulating a flexible and scalable simulation platform for modelling energy-aware cloud computing systems. The models exhibited presents the models for performing

This chapter presents our approach of a flexible and scalable simulation platform for modelling energy-aware cloud computing systems. The design of this simulation platform is aimed to build cloud models by configuring each part of the cloud system independently, that is, the hardware part of the cloud, management policies, virtualisation and energy consumption.

The new contributions presented in this chapter are propose a model for system architecture and propose a model for energy of system. On one hand, this design provides a cost-effective method to build a wide range of cloud configurations. On the other hand, large systems, containing a vast number of nodes, can be modeled in order to analyse the energy consumed by these nodes, and therefore, the total amount of energy required to supply the complete system. Moreover, iCanCloud has been designed to measure the energy consumption of each modelled hardware device, it being customisable for including new models, widen the possible scenarios built using iCanCloud. Therefore this allows to perform experiments with different level detail focused on different areas of the scope of distributed systems.

An autonomous energy management has been included jointly with hardware components. Thus, theoretical models from different researching studies has been incorporated into component models, measuring the energy consumed of each component. This technique avoid invasive or costly techniques to measure the energy consumed by hardware devices.

Moreover, virtualisation of physical resources is the main aim of cloud computing systems. The optimization of hardware resources usage in these systems involves saving energy, reducing the carbon footprint. Otherwise, it may entail to harm applications performance due to the servers overloaded. In order to ease the study of techniques for improving physical resources usage in cloud computing environments, next chapter *Simulating energy-aware virtual environments in cloud systems* exhibits the simulation models for virtualisation layer of the iCanCloud simulation platform.

Chapter 4

Simulating energy-aware virtual environments in cloud systems

This chapter presents the models for simulating the virtualisation layer in cloud computing environments. Basically, these models exhibits the management of tenant requests, network addresses, physical resources hostage, and how to obtain energy-efficiency measurements from physical devices. The inclusion of virtualization models into the simulation environment will allow to analyse existing/new scheduling techniques, improving servers performance, and saving energy from cloud computing systems.

The new contributions presented in this chapter are the techniques for modelling virtualisation in cloud computing environments, it being implemented in the iCanCloud simulation platform. Commonly, performing experiments on real hardware may be costly, invasive to the hardware and difficult to be fulfilled without harming users performance. This proposal is focused on providing techniques to perform studies in those elements that have a direct impact on both performance and energy efficiency of data centers, such as hypervisor scheduling policies, virtual machines management policies, and different user workloads.

4.1 Introduction

Cloud computing offers to users a pool of interconnected resources through the Internet, hindering the physical layer that supports the computational load. The usage of virtualisation techniques boost the growth of cloud computing, offering to both providers and users capabilities that are not possible at bare-metal computers. Thus, the main benefits for exploiting virtualisation in large systems, such as cloud systems, are summarised as follows:

- Accessibility: users data are always available to be accessed from different devices and any geographical area.
- Uptime increment of applications: live migration, storage migration, fault tolerance, and distributed resource scheduling techniques allow to increment applications uptime.

- Improvement of disaster recovery: by removing dependencies between applications and hardware vendors/server models, no longer maintaining identical even obsolete hardware in order to keep the production environment.

- Ease applications prototyping: due to the possibility of build self-contained labs into virtual machines, working over isolated networks.

Server virtualisation techniques allow to isolate applications into virtual machines. Thus, a single physical server using virtualisation techniques, permits several virtual machines allocation on it, even executing different operating systems. This technique is known as virtual machines management (in short, VMM). An example of VMM is server consolidation, which aims to reduce the energy consumed by servers and the overall footprint of the entire data center. Basically, it is focused on allocating as much virtual machines as servers can support, in order to reduce the number of active servers. Consequently, it requires less servers running, less networking gear, and less number of racks needed, being translated to less space required at a data center room. The final result is a reduction of monthly electricity bills, invested on supplying servers and cooling system of the data center. Unfortunately, once a server reaches its full operating capability, this causes a direct performance decrement on applications executed at hosted virtual machines. Therefore, archiving a balance between energy consumption reduction by using VMM techniques and performance reduction is a current challenge and hot topic.

Providers of public cloud computing environments offer virtualisation services to be purchased by users. The problem arises when thousands, or even millions of users, access concurrently to the system. It hampers system modifications due to the impact on performance and energy efficiency. On the one hand, a change at provisioning resources policy may have a positive impact by saving energy and improving the performance of servers, such as server consolidation techniques. On the other hand, applying different techniques on data centers to reach a fair balance between energy consumption and performance may be difficult, being conditioned to several factors. Some of these factors are: the workload variability of users, the hypervisor management techniques to virtualise physical devices of servers, the selection of servers from data center to host VMs, the quantity of VMs that are placed at each node, and the technique (commonly) costly/invasive to servers that is applied to obtain power consumption metrics.

In order to alleviate these factors, the simulation models for virtualising cloud computing environments have been designed. In addition, these models are included into iCanCloud simulation platform allowing to simulate virtual machines, hypervisors, VMM techniques, and the trade-offs between performance-energy impact by using different scheduling policies. Therefore, physical resources model previously described in Chapter 3 supports the virtualisation layer models.

Figure 4.1 shows a basic schema of virtual resources and management systems in iCanCloud. Due to the underlying dynamic nature of cloud environments, the core piece responsible for managing physical resources, interacting with tenants, and brokering virtual resources, is the cloud manager. It links tenants with physical data center resources. The tenants represented by the left box at figure, need the cloud manager to attend their requests for resources. Thus, cloud manager is responsible for attending a set of incoming petitions by following a previously set-up scheduling policy. In order to provide flexibility

and usability to the simulation models, these scheduling policies may be customized or new-designed by users, including them at simulation repository.

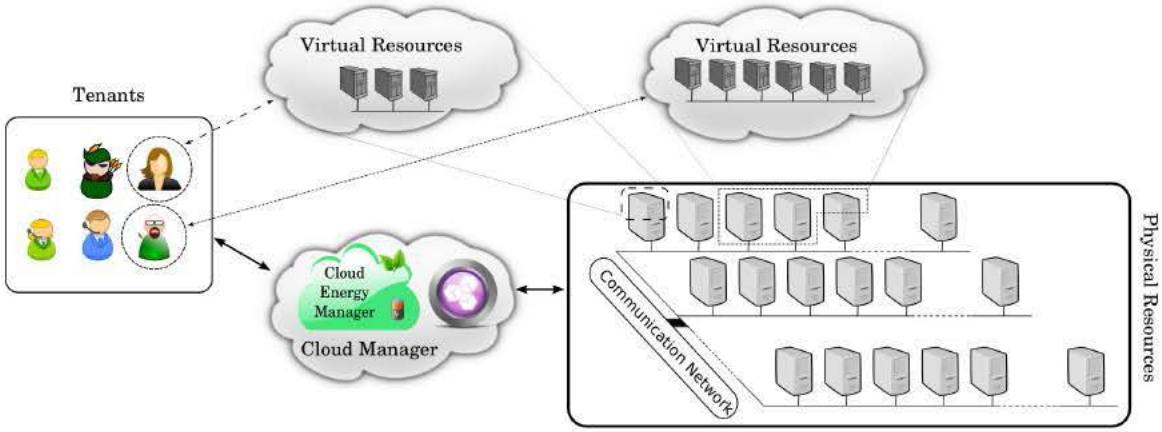


Figure 4.1: Global schema of iCanCloud virtualisation model

The *Energy Manager*, that is a part of *Cloud Manager*, manages the energy data of nodes. It links the node energy meters to obtain the power consumption of physical devices built-in each server.

The link between the *Cloud Manager* and the *Physical Resources* exhibits the responsibility for managing physical resources, creating *Virtual Resources* for the *Tenants* to interact with them. The *Virtual Resources* are: computing power, amount of storage, amount of memory, and bandwidth network, that can be purchased by *Tenants*. Therefore, these *Virtual Resources* correspond directly to virtual machine images allocation offered by cloud provider, and deployed at system by *Tenants*.

Next, iCanCloud models in charge of virtualisation layer of computing systems are shown, being hypervisor responsible for node physical resources management, and the cloud manager responsible for servers resources provisioning.

4.2 Modelling the hypervisor

Virtualisation techniques isolate physical resources from software. Applications, operating systems, and system services, are usually encapsulated inside virtual machines in order to get better physical resources usage. The abstraction level offered by virtualisation liberates software from being tied to specific hardware components. Thus, logical environments that composes virtualisation techniques are independent from underlying architectures, giving to applications performed at virtual machines the capability to be executed on a wide variety of computer systems.

Currently, multiple alternatives can be found for building the virtualisation layer such as: KVM, Microsoft Virtual Server and Virtual PC, User Mode Linux, VMWare, Xen, Virtual Iron and Parallel Workstation. In spite of the great variety of software aimed to virtualise physical resources, the main concept of all these alternatives is to manage the applications requirements to perform operations, respecting the isolation imposed by virtual machines. Hence, the model for simulating virtualization is inspired by these al-

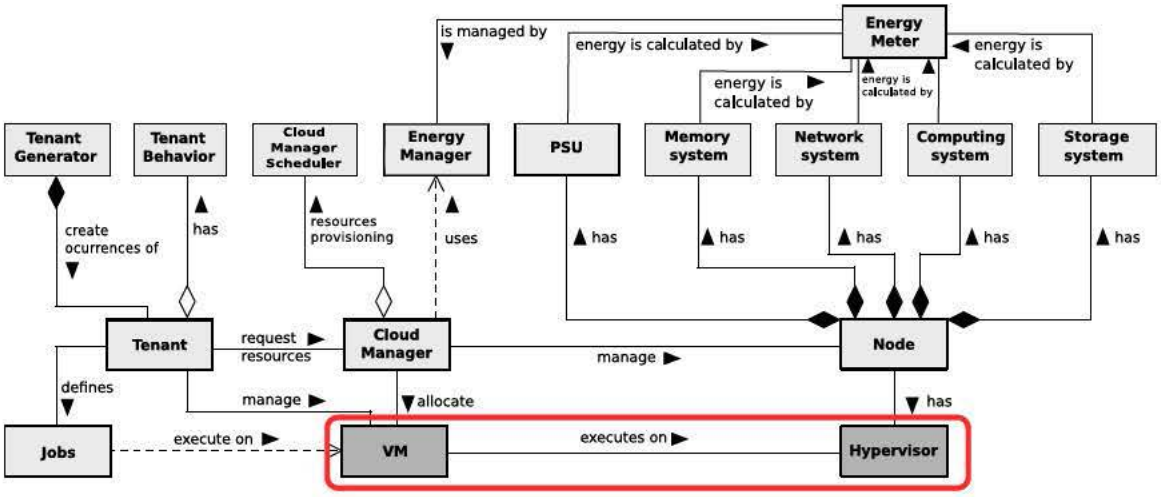


Figure 4.2: Class diagram focusing on virtualisation layer of iCanCloud

ternatives. This model achieves the following tasks: to orchestrate accesses to hardware devices by using a fully customized policy, and to offer a set of virtualized resources to perform applications over physical resources.

Figure 4.2 highlights with dark grey boxes the virtualization layer of the iCancloud class diagram. The *VM* box represents single virtual machine images requested by tenants. Moreover, the logic for managing the physical resources requested by VMs is responsibility of the *Hypervisor*.

Virtual machines accesses are managed by Hypervisor, that allows multiple tenant application instances defined as *Jobs* at Figure 4.2. These applications run concurrently within virtual machines on each single computer. The resources are dynamically partitioned sharing available physical hardware such as: CPU, storage, memory, and network devices. Each *Node* model has been designed to support the virtualisation layer management performed by hypervisor.

The *Hypervisor* has been modeled as special type of application. It is a container of applications, more precisely virtual machines. Main responsibilities for hypervisors are managing available physical resources, creating new virtual machines, releasing resources when virtual machines has finished, and scheduling operations performed by jobs running into virtual machines.

Figure 4.3 represents the internal structure of a node from a virtualisation perspective. The bottom area represents the physical resources: CPUs, memory, hard disk drives and network cards, that are described previously at Section 3.2.

Middle area of schema shows the virtualization layer, being the hypervisor the core of the model because it controls all the accesses to physical devices. All the operations issued by applications has to be scheduled passing through the hypervisor. Four parts composes on the hypervisor model: the storage manager, the network manager, the memory manager, and the CPU manager. Each manager handles a specific type of operations, depending on the resources that are requested by applications: the storage manager controls instructions issued to the storage devices; the network manager administers the communication between nodes; the memory manager supervises the input/output operations to main memory de-

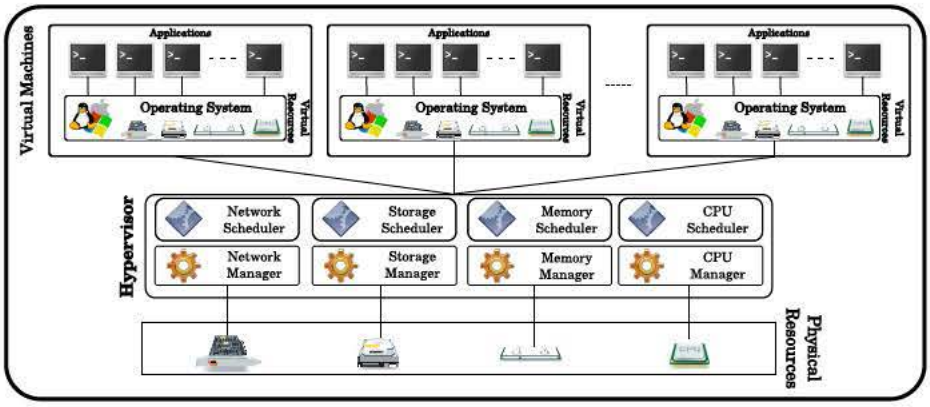


Figure 4.3: Global schema of the virtualisation model in iCanCloud

vice; finally, the CPU manager controls requests to perform computational accesses. These managers handle the amount of resources previously assigned to virtual machines are issued by requests. In order to control physical resources that have been assigned to virtual machines, each manager is specialized by a scheduler. These schedulers are responsible for priorising the accesses from virtual machine requests to the hardware layer. When several requests arrive at same manager simultaneously, the hypervisor selects the request or requests that will be performed by it scheduling policy. The requests that are not selected by scheduler, are enqueued until enough physical resources will be released.

Virtual machines are at the top layer of the virtualisation schema. A virtual machine is a set of physical resources purchased by tenants to launch applications. Currently applications and operative systems have been designed to run directly on bare-metal hardware, assuming that they fully 'own' the computer hardware. This schema has been adapted to the virtual machine simulation model, isolating user applications and its operative systems from node internals. Virtual machine model provides to tenants the total functionality of a complete computing node, but lacking of physical resources. Therefore, the operating system of virtual machines is mapped directly to the hypervisor. This entails that when a tenant application performs a request for resources, it passes through the operating system of the virtual machine, being redirected to a concrete hypervisor manager depending of the physical devices demanded by application. From tenants and applications perspective, when a request passes through the operating system it accesses directly to the physical resources. However, virtual machines only see the amount of physical resources that were assigned by hypevisor.

Users may create and customize hypervisor scheduling policies. Besides, manager internals and the methodology to create new management policies for node devices is described to each hypervisor manager.

4.2.1 Modelling hypervisor CPU management

The hypervisor CPU management is responsible for selecting a CPU core to execute incoming compute requests from tenant applications. It is divided in two parts: hypervisor CPU manager and hypervisor CPU scheduler. While hypervisor CPU manager is responsible for the background of the CPU management, performing tasks such as to separate

(prioritise) between tenants and system requests, the CPU scheduler may be modeled using the hypervisor API, handling tenant application requests, and managing the CPU cores states. The basic idea to supply flexibility to the hypervisor model, consists on providing an API allowing to users modeling existent scheduling strategies, and creating new ones. Figure 4.4 shows the set of methods provided by hypervisor CPU scheduler. For the sake of clarity, only the most relevant functions are listed.

```
1 int       hypervisor_cores_quantity();
2 long int  hypervisor_get_maxSpeed();
3 long int  hypervisor_get_currentSpeed(int coreIndex);
4 int       hypervisor_number_of_states();
5 int       hypervisor_get_core_state(int coreIndex);
6 double    hypervisor_get_cpu_power();
7 double    hypervisor_get_cpu_energy();
8 void      hypervisor_core_change_state(int coreIndex, int statePosition);
```

Figure 4.4: Hypervisor CPU scheduling API

Operations from this API can be grouped by two subsets. The first one groups operations to check CPU physical features from processing unit:

- *hypervisor_ cores_ quantity* obtains the total number of cores that build-in the control processing unit. This information allows to calculate which will be the core selected by the scheduling policy to perform a computational request.
- *hypervisor_ get_ maxSpeed* needs to receive which core is the focus of the operation. It returns the maximum speed that the core can reach to perform computational operations. The maximum speed of a core is measured as the highest number of Instructions Per Tick (in short, IPT) that a core is able to execute at 100% of its computing power.
- *hypervisor_ get_ currentSpeed* needs a core as parameter. It returns the current IPTs that selected core is able to perform.

The second group is designed to check CPU energy measurements, performance states, historical data of energy consumed, and to change the state of the core.

- *hypervisor_ number_ of_ states* allows to know the performance states that a core may reach to execute a computational request. Therefore, the states of a core can be modified or completely redefined by users, modeling their own energy features for different models of CPU, as it is described at Section 3.3.
- *hypervisor_ get_ core_ state* needs the core that will be the focus of the operation. It returns the performance state position at energy states structure. This information allows to the scheduler to perform decisions depending of the power consumption that the system is supplying to the core.
- *hypervisor_ cores_ power* and *hypervisor_ cores_ energy* are connected to the energy meter. These methods allow to know the energy measurements from CPU devices. The first operation allows to know the instant power that the CPU is wasting to feed the device. Moreover, second one obtains the amount of energy that CPU have been consumed from the beginning of a simulation experiment.

- *hypervisor_ core_ changeState* changes the state of the core *coreIndex*. This operation is the unique that modify the state of the CPU. It needs the core index that is the focus of the operation, and the state that will be its new performance state. Moreover, the performance state of a core may be modified by CPU scheduler depending on the workload. Therefore it is scheduler responsibility to select the correct states if a change is performed.

In order to exhibit how to implement a scheduling policy for virtualising the CPU, besides two different strategies for managing CPU requests in a virtualized node are showed. Concretely, the credit strategy and pools of CPUs. Both strategies has been modeled using the CPU hypervisor API.

Modelling the credit based CPU policy

The credit scheduler is CPU scheduler designed to minimize the CPU sharing fairly between requests for virtual resources proposed by Xen Project [29]. Xen Project is an enterprise that provides the open-source virtualisation solution Xen hypervisor, including the credit CPU scheduler as scheduling policy. Xen hypervisor is capable for virtualising ARM, x86-64, and x86 instructions, running over standard guest OS like Linux, Windows, and Solaris.

Credit CPU policy has been designed to minimize the wasted CPU time by having computing resources always working. The scheduler defines an abstraction level for the virtual resources requested by virtual machines as virtual CPUs (vCPU in short). Each virtual machine is owner of a vCPU that are managed by physical CPUs (pCPU in short). The pCPUs structure consists on a source of computing power and a local run queue of runnable vCPUs sorted by priority. Those priorities are sorted by Domains. Furthermore, between all domains, it exists a management domain, also known as driver domain or Domain 0, depicting basically a virtual machine with high level permissions accessing the hypervisor. It can perform extra operations such as the creation, destruction, and configuration of virtual machines.

Xen credit CPU policy has been modeled by three main states: when a request for resources arrives at CPU manager (see Algorithm 1), when a request has been computed and it returns from the CPU (see Algorithm 2), and when credits are burned and a compute request is sent to the CPU (see Algorithm 3).

Algorithm 1 exhibits the main structures creation and management due to the arrival of an application request for computing resources. The first step that credit scheduler performs is to obtain information about the domain associated to the arrival (line 6). If it does not exists, the domain structure is created by scheduler. Most remarkable details of this step is the assignation of parameters cap and weight for the new domain. The credit scheduler assigns to each domain a weight, and optionally a cap. The weight is a value for the relative CPU allocation of a domain; a weight value of 512 will receive as much CPU time as a domain with the default weight of 256. The cap parameter sets the absolute amount of time limit that a domain can receive. It is expressed in *hundredths* of a CPU which can be upper than 100 for multiprocessors. The importance of both values is due to scheduler uses the weight and cap parameters to obtain the amount of credits for each vCPU. Hence when a vCPU is executing, it is consuming credits.

Subsequently to the domain creation, the set of vCPUs are created for the correspond-

Algorithm 1 CPU Credit based policy - Compute request from application

```
Require: Cores hooked as pCPU structure.
Ensure: Each core has an initialised local run queue.
 1: Let                    hypervisor_cores_maxSpeed()
 2: Let                hypervisor_cores_quantity()
 3: Let            message.getVmId()
 4: Let              message.getUserId()
 5: if (                          ) then
 6:    if (  existsDomain(          )) then
 7:                        getNumberCPUs(       )
 8:                        vCPUSpeed(       )
 9:         Create the Domain structure
10:                    createDomain(       )
11:                .credit     (-(1«30))
12:                .cap     (————————
13:                .weight     256
14:       for (       to              ) do
15:                   .name     (       )
16:                   .Domain
17:                   .tSlice     30ms
18:                   .lastUpdate     (-1)
19:                   .pCPU     get_pCPU_with_less_domains()
20:                   .Priority     IDLE
21:          insert(          ,          )
22:       end for
23:       insert(        ,          )
24:    end if
25:
26:    if (          ) then
27:       if (      .credit     (-<(1«30))) then
28:                .credit           .weight
29:       end if
30:              .tSlice     30ms
31:       if (       .Priority == BOOST) then
32:                 .Priority     UNDER
33:       else
34:                 .Priority     BOOST
35:       end if
36:    else
37:                  vCPUminLoad(       )
38:    end if
39:    enqueue(          ,       )
40:    insertOrderedRunQ(        ,        )
41: end if
```

ing virtual machine (line 14). The vCPU needs a name and domain parameters to identify it, and a set of parameters to know its state in order to prioritize operations between virtual machines requests.

vCPU priority value is IDLE (line 20) due to the simulator management structure. Simulation model requires a state that represents the management phase for the vCPU recently defined, and waiting to execute a CPU load message. The priorities established by Xen credit scheduler are classified into active and non-active. Actives priorities can be OVER and UNDER, representing a vCPU that do not exceeded the previous assigned share of CPU resources (tSlice parameter) by period. Otherwise, when a non-active vCPU

is executed, it is inserted at run queue of a pCPU setting its priority to BOOST, which maintains it as the head of the run queue, over vCPUs with priorities of UNDER and OVER (line 31).

Once the Domain and vCPUs are created, the incoming request is enqueued at pending messages of vCPU(line 39), and the vCPU is enqueued into a pCPU queue ordered by credit(line 40).

Second state of credit scheduler models a computed request arrival from the CPU to the hypervisor (see Algorithm 2). This phase shows the scheduling decisions performed depending of the availability of the credits, the time slice, or the incoming message completion.

Algorithm 2 CPU Credit based policy - Message arrival from CPU

```
1: if (                                  ) then
2:    Let          get_vCPU(          .getVmId())
3:    Let            getDomain (          .getUserId())
4:    Let          identifyCore(          )
5:                    false
6:    if (finishCompute(          )) then
7:       hypervisor_return_message(          )
8:       if (  hasPendingMessages(        )) then
9:          if (       .pCPU            ) then
10:            migrate_vCPU(       ,       .pCPU)
11:         end if
12:                .Priority
13:         insertToIdle_vCPU(       )
14:           Change the state of the Domain if all vCPUs are IDLE
15:         if (                    ,       .Priority =         ) then
16:            migrate_vCPU(       ,       .pCPU)
17:         end if
18:      end if
19:   else
20:      burnCredit(       ,          )
21:      if (         .credit   0) then
22:                  .credit    (-1)
23:               .Priority    OVER
24:      end if
25:      if (          0) then
26:         enqueue(          ,       )
27:         setMessageToExecute(       , getMessageHead(       ))
28:      end if
29:            .tSlice    30ms
30:      if (       .Priority == BOOST) then
31:               .Priority    UNDER
32:      end if
33:      pushBackRunQ(       ,       )
34:      execute_pCPU_head(getMessageHead(       )
35:   end if
36: end if
```

First action consists in set the pCPU that has computed the message as free at line 5. Afterwards, there are two possibilities depending if the message has totally finished, or if it has consumed the time slice assigned by scheduler to be performed. In the case that the request has been completely performed, the scheduler returns the execution results to

the application that demanded for computing services (see line 7). Therefore, if vCPU has pending messages, the process to restructure vCPUs priorities is initiated before a new request will be chosen to be computed.

Moreover, if the message has been dislodged before its completion, the scheduler initiates a process to burn credits (see line 20). It is described at third phase, Algorithm 3. In addition, if the vCPU wasted all its credits, its priority turns into *OVER* state.

Otherwise, if message slice is over, it is enqueued at last of vCPU messages. Subsequently, new message selection to be computed is the head of vCPU queue (see line 27). Once a vCPU is scheduled, it receives a new time slice of 30ms.

Finally, the vCPU owner of incoming message is positioned at last position of pCPU running queue (line 33). Therefore, the head of pCPU is chosen to perform one of its messages. As a consequence, if there is more real CPU available than the demanding user domains, all domains get all the CPU they want. When there is contention, which means that the domains in aggregate want more CPU than actually exists, the scheduler arbitrates fairly between the domains that requires CPU.

The Algorithm 3 exhibits four main processes internals: the credits burning (line 1), the migration of vCPUs from a busy pCPU to a free pCPU (line 14), and finally the migration process to return a vCPU to its original pCPU (line 22).

The pCPU run queues are sorted by domain priorities, in fact vCPU priority rather than credit. On each pCPU, no matter what scheduling decision is made during the common path of the scheduler, the next vCPU is picked off from the head of the run queue. As a vCPU runs, it consumes credits of 100 every 10ms (line 11).

If a pCPU does not find a vCPU priority *UNDER* at its local run queue, a vCPU allocated into other pCPU run queue is migrated to be executed on it (see line 14). This is a mechanism for load balancing that guarantees a fair share of computing resources to all virtual machines. Moreover, when a pCPU changes its state to *IDLE*, the scheduling policy checks the local running queues from other pCPUs. If there is an overloaded vCPU, it will be migrated to a pCPU local run queue at idle state, as it is shown at line 18. This avoid the possibility of maintain a pCPU at *IDLE* state while may be runnable work waiting for executing at system.

Furthermore, when a pCPU reach *IDLE* state, a process to return the vCPUs to their initial pCPUs begins (line 22). When a vCPU is at *OVER* state, and its pCPU allocation is not the pCPU initial allocation, it is moved to the pCPU owner of it. This action allows to equilibrate the system as the original scheduling calculations, avoiding extra operations to recalculate new allocations of vCPUs, dealing with situations like an overhead of incoming messages.

When the time slice of a running vCPU expires, if this vCPU is still runnable, it will be put after all other vCPUs of equal priority to it (see lines 20 and 29).

Modelling the pools of CPU policy

Xen 4.2 offers a scheduling policy known as CPU pools. The main idea of this policy consists in grouping the processing units as different pools. These pools may be configured by users, modifying a set of configuration parameters to define their own scheduling policies. Moreover, logical CPU allocation into a concrete pool is performed dynamically. Therefore,

Algorithm 3 CPU Credit based policy - Burn credits and execute

```
 1: Let [          ]   empty queue to allocate idle and off cores.
 2: Let [          ]   queue to allocate domains with more than one pending requests to execute.
 3:   Burn credits every 10ms
 4: for (      to          ) do
 5:             get_pCPU( )
 6:     Get idle structures at pCPU and overloaded cores due to vCPUs.
 7:   if ((hypervisor_get_core_state( ) == IDLE)   (hypervisor_get_core_state( ) == OFF)) then
 8:     [          ]
 9:   else
10:               getExecuting_vCPU(      )
11:     burnCredit(      ,      )
12:   end if
13: end for
14:   migrate vCPUs overloaded to pCPU queues at idle state
15: while (([          ].size() > 0)   ([          ].size() > 0)) do
16:           [          ].pop()
17:           [          ].pop()
18:   migrate_vCPU(      ,      )
19:   setMessageToExecute(      , getMessageHead(      ))
20:   insertOrderedRunQ(      ,      )
21: end while
22:   return vCPUs to their initial pCPU
23: for (      to          ) do
24:   for all (          .runQueue) do
25:
26:     if (      .priority == OVER)   (          ) then
27:       migrate_vCPU(      ,      )
28:       setMessageToExecute(      , getMessageHead(      ))
29:       insertOrderedRunQ(      ,      )
30:     end if
31:   end for
32: end for
```

a virtual machine is only assigned to one pool, but it can be moved from one pool to other.

The pools of CPU policy offers accesses to an isolated number of CPUs by a set of VMs. In spite of providers may increment the price of rented resources using this scheduling model, clients have the guarantee for accessing an exclusive set of physical resources. This provides to users QoS benefits, having the possibility for selecting their own scheduling parameters such as the time-slice, rate-limit, or different scheduling policies for their hired resources. For example, a client with six virtual CPUs may rent the equivalent for two cores in order share its computing power only by his VMs.

When a user appears at system, a pool of CPUs is created to him. Thus, the set of CPUs that he paid for will be booked and located as his pool. Moreover, each pool scheduler only manage the cores at his pool. The model for the pools of cpu scheduling policy is presented at Algorithm 4.

Pools scheduling model has been divided into two main functionalities depending on the requirements of the arrival request. Both are the pools creation (line 9), and the actions performed when a request demanding for computational power arrives (line 35). The first one, the pools creation, needs to ensure that the number of requested pools can be supplied by physical system (see line 12). It is responsibility of the hypervisor to control the available

Algorithm 4 CPU pools scheduling policy

Require: A new operation pool Creation.

Ensure: When a pool creation is requested, it is responsibility of the hypervisor to control that there are at least the resources requested by tenant

```
 1: Let                hypervisor_cores_quantity()
 2: Let              Empty vector for allocate users pools.
 3: Let                Operation from arrival message
 4: Let                 Tenant ID from arrival message
 5: Let               Requested physical cores from arrival message
 6: Let                Pool created to allocate physical resources, and tenantID for managing the resources
    from one tenant.
 7: Let          Variable to allocate the cores at IDLE or OFF state
    Message Arrival
 8:                          .getOperation()
 9: if (               poolCreation) then
10:                            .getUserId()
11:                           .getOperationSize()
12:    if (                      ) then
13:                .setResult(       _          _              )
14:    else
15:                       createPool(          )
16:       for (      to           ,             ) do
17:          if (             = unassigned) then
18:
19:                                  - 1
20:
21:          end if
22:       end for
23:       if (                        ) then
24:
25:       else
26:
27:       end if
28:         Configuring internal sub-scheduler using message parameters from tenant.
29:                .SchedPolicy    getSchedPolicy(message)
30:                .SchedParameters    getSchedParameters(message)
31:               .setResult(       _          _     )
32:    end if
33:    hypervisor_return_message(          )
34: else
35:       Incoming message request for compute.
36:                           .getUserId()
37:                         .getVmId()
38:          0
39:    for ( =              to          ,          .tenantID              ) do
40:                    get_vCores(          .vCores, vmID)
41:                   subSchedule(          ,          ,          )
42:                      true
43:       hypervisor_core_compute(          ,          )
44:    end for
45:    if (              ) then
46:                .setResult(          )
47:       hypervisor_return_message(          )
48:    end if
49: end if
```

physical cores in order to distribute the computational power.

Afterwards, the pools creation begins by the pools selection, grouping the unassigned cores to pick up the quantity requested by tenant (see line 16). The number of requested cores are allocated in a structure that is identified by using the tenant unique identification (*tenantID*) to his pool. Besides, the core is set that it owns to the tenant (line 20).

Moreover, if the tenant could have any other pool at same node, the new pool is joined to previously purchased pools at preceding scheduling executions (line 23).

Finally, last actions performed to create a pool consists on the scheduling policy configuration. The user may choose his own scheduling policy by defining some parameters obtained from the incoming message (line 28).

Second functionality consists in processing a message arrival that demands for computational resources. The aim of this case is to find the tenant pool between the existent ones, selecting one core to execute the request (line 39). This selection depends of the selected policy defined by user. It is represented at algorithm by *subSchedule* operation at line 41. Note that the function *subSchedule* mask the tenant scheduling policy for his pool. Therefore this scheduling policy has to choose the core where the message will be computed.

4.2.2 Modelling hypervisor memory management

The memory virtualised by hypervisor involves sharing the physical system memory by dynamically allocation of virtual machines requirements. Hence it is necessary to ensure that each virtual machine will have the amount of memory it requires, guaranteeing a correct functioning of the virtualization layer. The hypervisor is the responsible for providing to each VM an adequate amount of memory. But it is also responsible to do not assign more memory to a VM than it really needs. An excessive memory reservation for each VM limits directly the quantity of VMs that hypervisor is able to allocate on the same server, and it can even have a negative impact on its performance.

Alternatively, the physical memory defined by VM image is not entirely available to its usage. The memory management model include elements based on the overhead due to the virtualization layer. The memory overhead model may be customized by users depending on real hypervisor models. It is composed by the amount of memory needed by hypervisor software to be run at system, that will be different depending on the deployed services; the amount of memory needed for hosting the operative system at root partition, that hypervisor may use for its own application management, allowing the possibility to run extra applications or extend VM memory in the parent partition; and finally an amount of memory representing the overhead that a VM needs to be deployed at system. Each VM performed in a server needs to reserve a small quantity of memory representing integration services and virtualization-processes. This is about 32MB of memory each GB of RAM that a virtual machine image requires to be allocated by hypervisor.

The level detail of the hypervisor memory virtualization may be customised by users in order to create new management models or modifying existing ones. Therefore, the hypevisor memory API has been modeled for managing memory operations. Most relevant methods are listed at Figure 4.5:

Operations from this API can be grouped by two subsets. First subset groups opera-

```
1 double hypervisor_get_total_physical_memory();
2 double hypervisor_get_available_physical_memory();
3 double hypervisor_get_total_VM_memory(int vmID);
4 void   hypervisor_set_total_VM_memory(int vmID, double newAmount);
5 double hypervisor_get_total_VM_memory(int vmID);
6 void   hypervisor_set_available_VM_memory(int vmID, double newAmount);
7 double hypervisor_memory_power();
8 double hypervisor_memory_energy();
```

Figure 4.5: Hypervisor memory scheduling API

tions to check physical features of memory device. These operations are described before:

- *hypervisor_get_total_physical_memory* returns the total amount of memory (in Mega Bytes) that the physical server has built-in.
- *hypervisor_get_available_physical_memory* obtains the quantity of available memory from the physical memory system of the node. This operation is allows to calculate if a VM may be deployed at the server. This operation has not setter associated due to the information is extracted directly from the physical memory model. It is physical memory model responsibility to manage correctly the available physical memory.
- *hypervisor_get_total_VM_memory* operation is analog to obtain the physical memory but directly oriented to a VM. Due to this fact, the identifier from the VM to obtain the result is required.
- *hypervisor_set_total_VM_memory* sets the maximum amount of physical memory that the VM with identifier *vmID* has available to perform IO operations.
- *hypervisor_get_available_VM_memory* returns the amount of free memory that a VM has for performing operations when the operation is invoked. It is required the VM identifier to locate the VM from hypervisor structures.
- *hypervisor_set_available_VM_memory* sets the total quantity of physical memory reserved for a VM. This operation does not set an amount of memory major then the total size of the memory defined at VM image. It is commonly used when IO operations have been performed.

Moreover, the hypervisor needs to manage directly the VMs memory usage patterns. Those parameters are collected into *memoryCells* structure. Each memory cell collects the virtual machine usage of the physical memory. The model of the basic memory cell allocates an identifier for VM, the total physical memory defined by VM image, and the amount of memory available for a VM. But memory cell model can be extended by users in order perform new memory scheduling policies. Therefore, operations from the memory API that actuate directly on memory settings of virtual machines use memory cell structures.

The second group has been modeled to obtain memory power consumption, and historical data of energy consumed by this device.

- *hypervisor_memory_power* performs a request to the energy meter in order to obtain the power that memory modules are consuming at a concrete instant.

- *hypervisor_ memory_ energy* obtains the total energy wasted at system server by the physical memory system. This value is obtained from the energy meter, that maintains the historical data of the energy consumed by memory modules.

Two techniques have been modeled to hypervisor memory management: static memory and dynamic memory. Both simulation models have been performed using the hypervisor memory API. They are exhibited at Algorithm 5, and Algorithm 6 respectively.

Modelling static memory management

The static memory model defines an amount of memory for each virtual machine. It ensures the amount of memory defined by VM image for each VM, but it does not allow to resize the quantity of memory depending on the requirements of the applications. This entails the impossibility oversize the quantity of hosted VMs, which is limited by the amount of physical memory of the server. Therefore, when a server is full, if hypervisor try to allocate new VMs, this process fails returning an *out of memory* error. There is no possibility to resize the VMs memory allocation. The only solution to reduce the existing VM memory allocation, consists in edit the memory features of each VM, save its state, and reboot the server.

The simulation model of hypervisor memory static management is exhibited at Algorithm 5. It has been divided into three main functionalities depending on requirements of incoming requests: to set a new VM at system (line 6), to erase a VM from the system (line 19), and finally to perform an I/O operation (line 26). The first one, when a virtual machine attempts to be linked to the system, the model checks if the physical memory has enough capacity to allocate the new VM. Therefore, if the VM allocation process begins, an amount of memory is demanded due to the requirements for virtualising services of the VM (line 10). As consequence, a new message is sent to the physical memory system in order to perform the allocation of resources. Besides, memory model sets the operation results into the incoming message. The message is returned to the demanding entity notifying the operation finalization status.

The second case, when the arrival message is related to a VM unset operation (see line 19), the memory model releases the amount of memory that virtual machine is using from physical system. Afterwards, internal memory structure is erased and initial message returned to its owner.

Finally, if an I/O operation is requested (see line 26), it could be a memory release operation or a memory allocate operation. If the operation consists on allocating an amount of memory, the memory available of the VM is decreased. By contrary, if it is required to release memory from the physical system, the memory available of the VM is increased the amount of memory requested by incoming message, before to send it to the physical resources.

Modelling dynamic memory management

The goal of cloud computing systems is to maximize the physical resources utilization. This impact directly on the memory system, lowering the margins between available physical

Algorithm 5 Hypervisor static memory model

```
Require: A message arrival (     )
 1: Let                    hypervisor_get_available_physical_memory()
 2: Let                    = 32MB per GB of predefined memory reservation for virtualization services
 3:              .getVmId ()
 4:                  .getOperation()
 5:               .getOperationSize()
 6: if (                    _       ) then
 7:    if (                 >       ) then
 8:       hypervisor_set_total_VM_memory(      ,       )
 9:       hypervisor_set_available_VM_memory(      ,       -                  )
10:         allocate an amount of memory for virtualized resources
11:                  .setOperation(          _              )
12:                  .setOperationSize(                     )
13:       hypervisor_send_to_memory(              )
14:           .setResult(      _      _     )
15:    else
16:           .setResult(      _      _     )
17:    end if
18:    hypervisor_return_message(     )
19: else if (                          _          ) then
20:                  .setOperation(          _              )
21:              (hypervisor_get_total_VM_memory(      ) - hypervisor_get_available_VM_memory(      ))
22:                 .setOperationSize(       )
23:    hypervisor_send_to_memory(              )
24:    hypervisor_unset_vm(      );
25:    hypervisor_return_message(     )
26: else
27:                      hypervisor_get_available_VM_memory(      )
28:    if (                          _              ) then
29:       hypervisor_set_available_VM_memory(      ,              -       )
30:       hypervisor_send_to_memory(     )
31:    else if (                          _              ) then
32:       if (                >       ) then
33:          hypervisor_set_available_VM_memory(      ,              +       )
34:          hypervisor_send_to_memory(     )
35:       else
36:             .setResult(      _      _     )
37:          hypervisor_return_message(     )
38:       end if
39:    end if
40: end if
```

memory and virtualized memory. Therefore, techniques like memory over-commitment is still a fundamental feature of virtualized infrastructures.

Dynamic memory technique lets to VMs to consume memory dynamically depending on performing applications. They become dynamic workloads which require to be handled by the cloud system in order to deliver the best application performance. Moreover, dynamic workloads creates trade-offs between performance-density of VMs. In order to improve this trade-off the dynamic memory management, also defined as *dynamic memory optimization*, *memory overcommit*, or *memory ballooning*, is adopted by hypervisors. The hypervisor automatically adjust the memory of running VMs, maintaining an static memory range that guarantees the performance, and allowing a high density of VMs per

server. The dynamic memory management technique has been modeled for the hypervisor memory manager, as it is exhibited in Algorithms 6 and 7.

Algorithm 6 Hypervisor dynamic memory model - set and unset VM cases

```
Require: A message arrival
 1: Let          = An extended memory cell to control logical and physical VM data
 2: Let               = Vector to allocate VMs memory usage data are allocated
 3: Let                 hypervisor_get_available_physical_memory()
 4:              .getVmId ()
 5:                   .getOperation()
 6:              .getOperationSize()
 7: if (                    _      ) then
 8:                calculate_logical_minimum(     )
 9:    if (                 >          ) then
10:                     .id
11:                     .logical_max
12:                     .logical_min
13:                     .physical_busy
14:                     .mem_buffer    0
15:                      .insert(               )
16:         allocate an amount of memory for virtualized resources
17:              .setOperation(          _              )
18:              .setOperationSize (                    )
19:       hypervisor_send_to_memory(         )
20:           .setResult(     _     _     )
21:    else
22:           .setResult(     _     _     )
23:    end if
24:    hypervisor_return_message(     )
25: else if (                    _           ) then
26:       free the amount of physical memory used by
27:                                      .search(     )
28:           .setOperation(          _            )
29:           .setOperationSize(               .physical_busy)
30:    hypervisor_send_to_memory(         )
31:                    .erase(              );
32:    hypervisor_return_message(     )
33: else
34:    ...
35: end if
```

Depending on the functionalities, dynamic memory model has been divided depending on the operation required by requests: the operation to set a new VM at hypervisor system (Algorithm 6, line 6), to erase a VM from the system (Algorithm 6, line 19), and finally to perform an I/O operation (referenced at Algorithm 6, line 33, but described at Algorithm 7). This model requires to extend the memory cell model to allocate the minimum logical memory size that the hypervisor must reserve for each VM, and a buffer size in order to control applications that request for more resources than they have assigned. The memory buffer size has to be defined by user at policy configuration. All extended parameters are shown from line 10 to line 14 where a new VM is set at system.

In the case where a virtual machine dislodged from a server, the scheduler operates as the static memory model. It is remarkable the fact that in the dynamic model, hypervisor calls have been unmasked in order to clarify the functioning of the extended memory cells.

Algorithm 7 Hypervisor static memory model - I/O operation case

```
 1:                                        .search(      )
 2: if (                          _                ) then
 3:                  .physical_busy     (                .physical_busy -      )
 4:    if (               .mem_buffer > 0) then
 5:                              (                .mem_buffer -      )
 6:       if                    < 0 then
 7:                          .mem_buffer     0
 8:       else
 9:                          .mem_buffer
10:       end if
11:       hypervisor_send_to_memory(          )
12:    end if
13: else if (                              _                ) then
14:                               .physical_busy +
15:    if (          <                .logical_min) then
16:                      .physical_busy
17:       hypervisor_send_to_memory(          )
18:    else if ((          >              .logical_min)   (          <              .logical_max)) then
19:       if (          _   _          _          _              >      ) then
20:                          .physical_busy
21:          hypervisor_send_to_memory(          )
22:       else
23:                                      searchOversizedVM(      )
24:          if (                        .size() > 0) then
25:             for (      to                          .size()) do
26:                 Free VMs memory buffers moving the memory to storage system
27:             end for
28:                              .physical_busy
29:             hypervisor_send_to_memory(          )
30:          else
31:                        .setResult(      _      _      _                )
32:             hypervisor_return_message(          )
33:          end if
34:       end if
35:    else
36:       if (          _   _          _          _              >      ) then
37:                          .physical_busy
38:          hypervisor_send_to_memory(message)
39:                          .mem_buffer     (              .physical_busy -              .logical_max)
40:       else
41:                     .setResult(      _      _      _                )
42:          hypervisor_return_message(          )
43:       end if
44:    end if
45: end if
```

The I/O operation case has been modeled at Algorithm 7. It has been divided depending on requested operation. If incoming message requests for a releasing memory operation (line 2), it is necessary to know if the VM has applications using extra memory. In order to test this, the memory cell is located and, if memory buffer is used, it is released from it (line 9). Otherwise, the memory to be released is picked up from logical VM memory. Finally, the message is sent to the memory system to perform the operation at physical memory.

When the operation from arriving message attempts to allocate an amount of memory data (line 13), it is necessary to know the state of VM memory. Hence, if it has enough space at its logical partition, the operation is performed directly. Note that lines 16, 20, 28 and 37 represents same hypervisor allocation operation, *hypervisor_get_available_VM_memory*. Moreover, if logical partition is full, it is necessary to know the state of physical memory.

Most remarkable state occurs when physical memory is full and there is not enough memory to allocate the requested amount (from line 22 to line 34). It is initiated a process to search over-sized VMs in order to obtain free memory from them, reducing their memory buffers by moving these amounts to the local storage system.

Finally, when the server has enough physical memory but the VM logical memory is completely full (line 36), the scheduling model uses dynamically the memory buffer for executing successfully the operation.

4.2.3 Modelling hypervisor storage management

The hypervisor storage manager virtualises storage resources of cloud computing environments, offering transparency to VMs. It provides uniform virtual storage devices and services required by tenants, improving the availability, speed, and utilization of data. A key for these improvements consists on enhance multiple disk storage systems management, including different models, by combining their individual capabilities with extended provisioning, data protection, replication, and performance acceleration services. Therefore, both transparency and the ability to combine multiple physical disks, enables to the storage hypervisor the benefit of hardware devices interchangeability. Storage hardware replacement and substitution from underlying architecture may be performed, without altering or interrupting virtual storage environment services.

The simulation model of hypervisor storage management allow to deploy existent models of storage which includes direct-attached storage (in short DAS), storage area network (SAN), network-attached storage (NAS), Unified Storage(SAN and NAS), and not existent models.

Figure 4.6 shows an overview of the hypervisor storage manager model. It has been simplified hindering the parameters of the operations in order to simplify the model. The figure exhibits two major parts described in the following: management/allocation of tenants storage data, and management of incoming messages.

The management of stored data by tenants is defined at *Storage Cell*. A storage cell is a structure modeled to control the data set that each VM is using, and to manage the locations of those data, offering transparency to tenants operations. There is a storage cell for each VM, and one for the root operative system of the server.

A storage cell manages the connections to local servers and to remote storage servers. If data from VM are not allocated at local server, the features of the connections are recorded using remote file storage structures. *Abstract_Remote_FS* is responsible for managing all IP connections, allocating pending requests before processing them. It offers the possibility to control a variable number of IP addresses destinations for storage servers. The quantity of IPs depends on the cloud model to storage the data.

Moreover, remote storage may divide the tenants data to be allocated at several storage servers. Therefore it is needed to model join and spread data operations depending on the

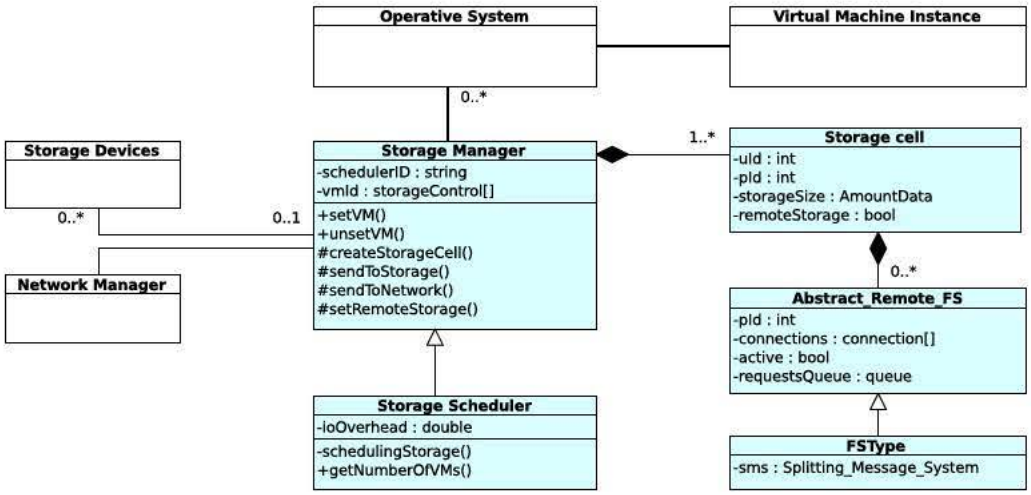


Figure 4.6: The hypervisor storage internals scheme

remote storage system. The file storage type (*FSType*) is in charge of these operations. It splits incoming data storage requests before sending them to the network. By opposite, it collects the corresponding messages from network that represents portions of data, until entire data requested by an application will be built. The *FSType* can be customized by users in order to study the different storage models.

The management of incoming messages is responsibility for the *Storage Manager* and the *Storage Scheduler*. These elements are linked to the VMs, concretely to the *Operative System*. When an application requests for storage resources, the process is managed by VM operative system internals. Besides, the request arrives to the hypervisor storage manager, where depending on its location, it may be redirected to local *Storage Devices*, or it may be sent the *Network Manager* in order to reach a remote storage server.

Storage Scheduler model prioritizes incoming tenant requests in order to perform different scheduling policies. These policies select which message execute depending on parameters like the amount of storage used by a tenant, or the power consumption of physical storage drives. In addition it is also responsible for managing the local storage devices, selecting the device where operations will be performed. The hypervisor storage manager provides an API to allow users modelling their own scheduling policies. Main operations of hypervisor storage API are exhibited at Figure 4.7.

```
1 int    hypervisor_getPhysicalDevicesQuantity ();
2 int    hypervisor_localStorageUtilization (string vmID);
3 int    hypervisor_remoteStorageUtilization (string vmID);
4 int    hypervisor_getFreeStorage (int deviceIndex);
5 int    hypervisor_getTotalSpace (int deviceIndex);
6 double hypervisor_get_storage_power (int deviceIndex);
7 double hypervisor_get_storage_energy (int deviceIndex);
8 void   hypervisor_send_request_toNIC (message* msg);
9 void   hypervisor_send_request_toDisk (message* msg, int deviceIndex);
```

Figure 4.7: Hypervisor storage scheduling API

Operations from this API can be grouped by three subsets depending if the goal of the operation is to obtain information from physical features, related to know VM storage usage, or aimed to operate with network or physical devices.

First subset groups operations to check physical features of storage devices. These operations are described before:

- *hypervisor_ getPhysicalDevicesQuantity* allows to obtain the number of physical storage devices that comprises the storage system of a server.
- *hypervisor_ getTotalSpace* returns the storage size that a VM has reserved to allocate tenant data. This quantity is defined by VM image and set into an storage control structure at Hypervisor Storage Manager.
- *hypervisor_ getFreeSpace* gets the amount of available space at storage system that a tenant may use.
- *hypervisor_ storage_ power* obtains from the energy meter the amount of power that a storage device is consuming when the operation is performed.
- *hypervisor_ storage_ energy* calculates the total amount of energy consumed by a storage device. This information is obtained from the energy meter.

The second group comprehends operations related to control the data generated by VMs at cloud storage system. These operations have been modeled to perform operations such as the usage costs due to the amount of data stored by a tenant, or the performance of storage system depending of its use. These operations are:

- *hypervisor_ localStorageUtilization* returns the amount of storage space used by VM to allocate data into the local server.
- *hypervisor_ remoteStorageUtilization* obtains the amount of storage used by a VM to allocate data into remote servers .

Finally, third set of operations groups the methods provided to redirect IO operations. The aim of the operations are not related to the physical device operations. They have been modeled for sending requests to different destinations:

- *hypervisor_ request_ toNIC* sends to the network manager a tenant request for performing IO operations into a remote storage server.
- *hypervisor_ request_ toDisk* sends the request to a selected physical storage device to perform IO operations. The index represents the local storage identifier where the operation will be performed.

The storage scheduler is responsible for managing physical storage devices of the server where hypervisor is running, and to supplying tenants requirements related to physical resources. In order to exhibit how a storage scheduler operates, a management strategy for operations such as a file creation, file writing, and file reading, are exhibited at Algorithm 8,

Algorithm 9, and Algorithm 10 respectively. These show a scheduling model that follows a first come first serve (FCFS) policy on tenant requests. Moreover all requests are performed to same physical device until it will not be able to allocate more data. This scheme has drawbacks on performance and reliability of data, but it has a lower energy consumption.

Algorithm 8 Hypervisor scheduling scheme - file creation model

Require: The node has several storage disks without distinction between them
Ensure: Each VM has reserved an amount of physical storage at creation process defined by VM image

```
 1: Let                    .getVmId()
 2: Let                     .getOperationSize()
 3: if (          .getOperation()                _          ) then
 4:    if (           .isLocalStorage()) then
 5:       if (hypervisor_getTotalSpace(       )) > (hypervisor_getFreeSpace(       ) +          ) then
 6:                            hypervisor_getPhysicalDevicesQuantity()
 7:                          getStorageCell(       )
 8:          for (     0 to [          ].size(),             ) do
 9:             if (                  _                      >          ) then
10:                hypervisor_storage_request(             ,           )
11:                                .file                .getFileName()
12:                                .KB
13:                                .StorageServers
14:                                .allocation     i
15:                             .fileControl.insert(                  )
16:                          = true
17:             end if
18:          end for
19:       else
20:                    .setResult(          _                 _          )
21:          hypervisor_return_message(            )
22:       end if
23:    else
24:       userInfo     getStorageCell(       )
25:                               hypervisor_getIPs (userInfo,            .getFileName())
26:       for (     0 to                        .size()) do
27:                                    .dup()
28:          hypervisor_request_toNIC(                )
29:       end for
30:    end if
31: end if
```

Algorithm 8 shows the scheduling scheme for a tenant file creation. When a message arrives at hypervisor storage manager, the scheduling model has different flows depending of its destination: local storage (line 4) or remote servers (line 23). If the file creation will be performed at local storage, the scheduler test if the virtual machine has enough space to allocate the new file (line 5). Otherwise, the hypervisor returns the incoming request to the demanding application, notifying that VM has not enough storage space available to perform the required operation (line 21).

In the case that the VM has available space to perform the operation, the scheduler searches the storage cell related to the VM that is performing the application to allocate the new file structure (line 7). A file structure allocates all data identification for a concrete file (lines from 11 to 15). These structures are managed by hypervisor storage manager controlling where file structures of tenant are located. The scheduling model selects first physical device that has enough free space to perform the application request.

If the creation of the file will be performed at remote servers (line 23), scheduler locate the IPs belonging to the data servers that hypervisor selects for storing files. These IPs are assigned by Cloud Manager Scheduler. Thus, depending on the cloud storage model, the quantity of data servers may be variable. Finally the incoming request is duplicated and sent to each data server (line 27). Each data storage server that allocates partially or totally a file needs to create an entry to identify it. This entry consists on file features such as: tenant owner, VM identifier from file was created, and unique file identifier.

Algorithm 9 Hypervisor scheduling scheme - file writing model

```
 1: if (          .Operation                _          ) then
 2:    userInfo     getStorageCell(          .getVmId())
 3:                          userInfo.fileControl.searchFile (          .getFileName())
 4:                                                .StorageServers
 5:    if (                      .size()      ) then
 6:                               userInfo.split (          )
 7:       split generates a numRequests   ————————————————————
 8:       for (     0 to                    .size()) do
 9:          hypervisor_request_toNIC(                  .popMessage())
10:       end for
11:       userInfo.pendingMessages                    .size()
12:    else
13:                                    .allocation
14:       hypervisor_storage_request(          ,          )
15:    end if
16: end if
```

Algorithm 9 reflects the flow of operations that the storage scheduler performs for a writing operation request. When an operation is identified as write file operation, the scheduler search the information related to the file that will be written (line 3). If the internal structure associated to the file contains a set of storage servers, it represents that the file was allocated into remote servers (line 5). Depending on the number of servers where the file was created, and the *FSType* model described at Figure 4.6, the size that will be written is splited to be sent by hypervisor to the network in order to reach the data storage servers.

The storage cells have a parameter to control the quantity of pending requests that has been sent to the servers. This feature allows to wait for the results of the operations before to notify to the demanding application the final result of the writing process. Moreover, when the number of requests generated to write a file is major than the number of servers, this scheduler has been modeled to control network congestion, avoiding to send to data server messages until the previous ones reach its destination.

Finally, if the file will be written at local devices, the file structure from storage cell contains the information to locate physical device index where the file was created (line 13). Besides, the hypervisor sends the request to the corresponding physical device.

Algorithm 10 describes the main scheme for a file reading operation. Firstly, the file information about its location is acquired to analyse if file was created at local storage devices or into a remote server (line 2). Depending if file entry has storage servers associated to it, the scheduler distinguish between local or remote allocation.

When a file has been located at remote storage servers (line 5), it is necessary to request the data to the servers where it was splited-up. To perform this task, the scheduler

Algorithm 10 Hypervisor scheduling scheme - file reading model

```
1: if (          .Operation                _          ) then
2:     userInfo     getStorageCell(            .getVmId())
3:                             userInfo.fileControl.searchFile (            .getFileName())
4:                                                .StorageServers
5:     if (                      .size()       ) then
6:                             .pendingFileSize                        .KB
7:         join waits until all requests that complete the file arrives where numRequests
8:         for (      0 to                      .size() - 1) do
9:             hypervisor_request_toNIC(             .dup())
10:        end for
11:        hypervisor_request_toNIC(             )
12:    else
13:                                        .allocation
14:        hypervisor_storage_request(             ,             )
15:    end if
16: end if
```

calculates the number of requests to obtain the complete file. The storage cell allocates each incoming message size associated to the file requested at *pendingFileSize*. This allows to control the completion of the operation (line 6). Finally, a copy of the original request is sent to each data server.

In the case that the file was created at local space (line 12), the scheduler obtains the physical storage index where it was stored. Besides incoming request is performed to this device for reading the file.

4.2.4 Modelling hypervisor network management

Network virtualization is a core part for cloud computing environments. A computer network begins with a NIC at host server. It is connected to a layer 2 (L2 in short) segment providing capabilities such as Ethernet and WiFi. L2 segments in turn are connected using switches, designing a L2 network which may be a part of a subnet layer 3 (in short L3) network. Several L3 networks are linked using routers, that consists on the key of the Internet. Commonly, a IT data center may have several L2 and L3 networks.

Each node has its own physical IP address (pIP in short) allowing to route the packages from tenant applications to the NIC in order to reach another server. In spite of server applications know the pIP addresses, in cloud computing environments the pIP addresses are hidden to tenants. Therefore when a VM is purchased by a tenant, a virtual IP address (vIP in short) is assigned to this VM, being different to pIP addresses. It entails the fact that routing packages from several VMs allocated at same server, each one having its own vIP, may be a complicated task to be performed. In addition applications running to different VMs at same node may provoke collisions opening same ports. It is needed to provide mechanisms to translate virtual IPs and ports to physical ones and vice-versa.

The network manager model has been designed to solve this drawbacks, providing an internal port translation and network address translation mechanisms. Figure 4.8 exhibits the scheme including all parts that consists of the network manager model.

One of main tasks that *Network Manager* performs is to schedule incoming messages from the *Operative System* of VMs to the *Network Interface Card*. When a tenant applica-

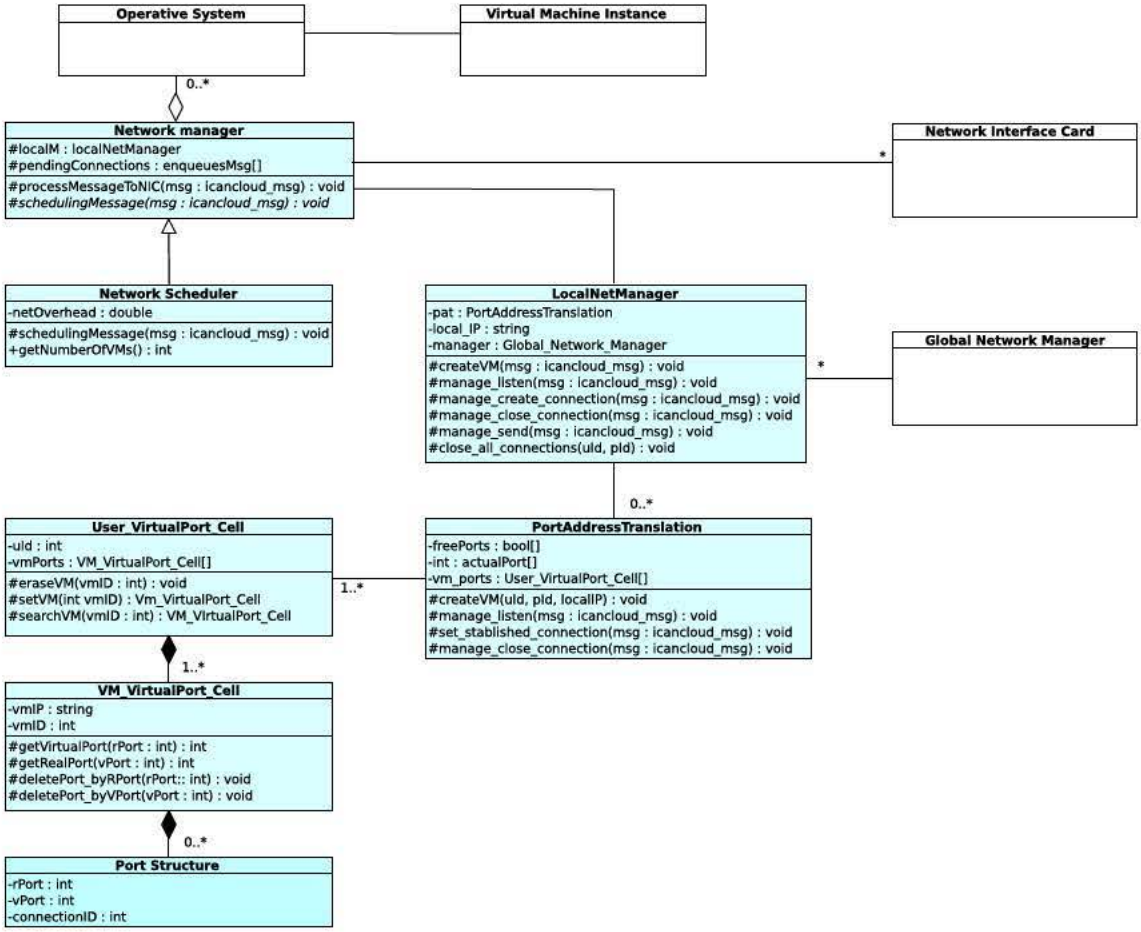


Figure 4.8: The hypervisor network internals scheme

tion performs a network operation, it arrives at *Network Manager* passing through the OS. Besides, OS redirects the request to the network manager, where it is stored at incoming queue messages. Thus, enqueued messages are managed using a policy defined by *Network Scheduler*.

Before a message will be sent to L2 segment, the manager invokes the *schedulingMessage* operation, responsible for prioritising incoming operations.

The API provided to schedule tenant messages is exhibited at Figure 4.9. It consists of operations related to obtain data from enqueued requests, to route messages, and to know energy measurements of NIC.

```
1 int         hypervisor_get_pending_requests_size ();
2 message*    hypervisor_get_incoming_request (int index);
3 void        hypervisor_send_request_to_net (message* msg);
4 void        hypervisor_send_request_to_hypervisor(message* msg);
5 double      hypervisor_NIC_power();
6 double      hypervisor_NIC_energy();
```

Figure 4.9: Hypervisor network scheduling API

- *hypervisor_ get_ pending_ requests_ size* allows to know the quantity of messages enqueued at pending requests queue of network manager.
- *hypervisor_ get_ incoming_ request* needs to receive an index that allows to locate a message at manager queue structure.
- *hypervisor_ send_ request_ to_ net* sends a message to the NIC in order to perform an operation into another server.
- *hypervisor_ send_ request_ to_ hypervisor* sends a message to the local hypervisor. This operation is provided for incoming requests from other nodes that aims to reach the local server.
- *hypervisor_ NIC_ power* returns the power consumed by NIC. This operation links the energy meter to obtain the energy measurements.
- *hypervisor_ NIC_ energy* allows to the manager to obtain the energy consumed by NIC. The energy consumed is obtained from energy meter, which maintains historical energy data consumed by NIC.

Moreover, main manager responsibility is for controlling and translating both virtual IP addresses and ports requested by VMs to physical ones, and vice-versa. This IP/Port translation allows to L2 and L3 layers to route messages, which only have the capability to manage physical IPs and ports. As it is shown at Figure 4.8, the model has an hierarchical structure managed by *Hypervisor Network Manager*, and beginning at *LocalNetManager*.

The *LocalNetManager* manages all network operations such as: listen, connect, send a package through network, and close a connection. In the cases that incoming requests from local VMs attempt to reach another VMs, it is needed to know the location of the server where the destination VM is hosted. In order to obtain this information, *LocalNetManager* uses the *Global Network Manager*, that belongs to Cloud Manager, being modeled to provide the physical IP where a VM is placed.

Furthermore, to solve collisions at ports due to the fact that several applications may request for using the same port, a *Port Addres Translation* (PAT in short) technique has been modeled. It controls the ports that VMs open and release. All information that PAT generates is supported by *User_ VirtualPort_ Cell* structures. Each cell allocates the set of VMs that a tenant has purchased, the virtual IPs, and the correspondence between each virtual port that applications request with the physical port that is assigned by PAT.

4.3 Modelling the resource provisioning

As desktop and server processing capacity has consistency increased year after year, virtualization has proved to be a powerful technology to simplify software development and testing, to enable server consolidation, and to enhance data center agility and business continuity. As it turns out, fully abstracting the operating system and applications from the hardware, and encapsulating them into portable virtual machines, has enabled virtual infrastructures features tolerant configurations on virtual infrastructure 24x7x365, with no downtime needed for backups or hardware maintenance.

Most virtualization efforts focus to maximize physical resources usage of data centers by using different techniques. Virtualization enables existing computer systems, optimizing data centers by reducing the number of physical machines that are needed to deploy, to manage, and to maintain applications. Regardless of underlying architectural features, minimizing capital expenses, operating costs, and energy consumption, it is important, the reliability, performance, and availability of the computing services required by clients, being the primary concerns of IT providers. Thus, techniques to balance the interests of both users and cloud providers are improved by scientific community, like consolidating multiple virtual servers on specific physical hardware platforms. These techniques allow to reduce the costs of physical resources, extending its life-time, and reducing energetic costs. Moreover, it is a hard task to find an optimal equilibrium for both sides. The deployment of new techniques into real data centers in order to improve the trade-offs performance-energy and efficiency-costs, may harm the QoS offered to the clients.

These trade-offs can be studied by using simulation techniques, without assuming risks to deploy new alternatives, neither investing efforts to implement new solutions. Therefore, the Cloud Manager has been designed for supporting and managing features to perform these studies. Next the model is presented in detail.

4.3.1 Cloud manager model

The Cloud Manager is composed by a set of components designed to allow the management of tenants, physical resources of computing servers/data storage servers, and evaluate existent and new resources provisioning policies.

The main aim for the Cloud Manager model is for attending the tenants requests. These requests are focused to purchase virtual resources as virtual machines from data centers. In order to perform the administration of the physical infrastructures resources, the Cloud Manager is designed following an hierarchical model in order to divide its main responsibilities. The global schema is shown at Figure 4.10.

The model has been divided into three parts depending of its responsibility: *tenants management model*, responsible for controlling tenants occurrences and its requests; *data center resources management model*, in charge of supply information about the features of devices, energy measurements, and usage of the servers; finally, *resources provisioning model*, allowing to schedule physical resources utilization depending of the requirements of the tenants, the energy consumption values, or the performance of applications.

Tenants management model

Figure 4.11 presents the main schema of tenants management by Cloud Manager. It is mainly composed by two sub-models: *Requests Manager*, and *Tenant Manager*. Only main operations are shown at figure for shake of clarity.

Tenants have total control to accomplish operations on their purchased VMs. However, there is a subset of operations that needs to perform processes on the physical layer of the data centers. These processes are encapsulated as *Requests*, being the tenants model designed for its control. The requests are grouped into three types, depending on the aim of the operation: *Cloud Request* related for purchasing virtual resources, such as the creation and release of VMs; *Storage Request* to perform operations on cloud storage;

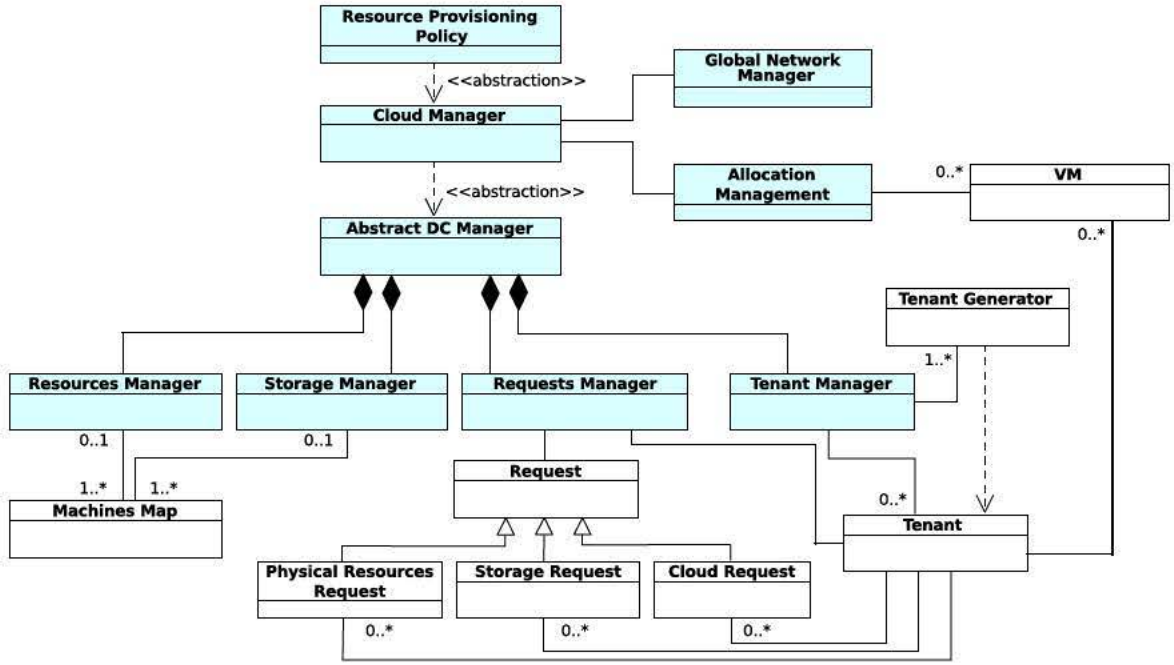


Figure 4.10: Global schema of the Cloud Manager model

finally *Physical Resources Request* to supply administration processes of system, launched by tenants-administrators role that aims for using the underlying architecture without virtualization. Nevertheless, the requests model is designed to be customized by users, supporting the creation of new types of requests, and the modification of the existent ones.

The *Requests Manager* model receives requests from tenants. These requests are inserted into an internal structure awaiting to be processed. It provides operations to obtain and classify the stored requests, in order to ease their selection by resources scheduling policy.

The *Tenant Generator* is in charge for creating tenants entities, simulating the real occurrences into a cloud computing environment. Thus, users may configure the total quantity of tenants that will be created, and statistical distributions defining the amount of time between tenant creations.

Once a tenant is created, it is registered by the *Tenant Manager*. This allows to notify the results related to the tenants requests when they are performed. Furthermore, if total number of tenants creations is reached during a simulation experiment, the *Tenant Manager* receives this information by the operation *generatorFinalization*, allowing to the Cloud Manager to decide if this condition is enough to finalize the experiment.

The layout of the Cloud Manager is the Abstract Data Center Manager model (at Figure 4.11 *Abstract DC Manager*). It controls main parts that composes on the underlying models. This scheme allows to build on it structures to perform different resources provisioning policies for computing systems, being cloud or cluster systems. *Tenant Manager* and *Requests Manager* are linked to it in order to be accessible by Cloud Manager.

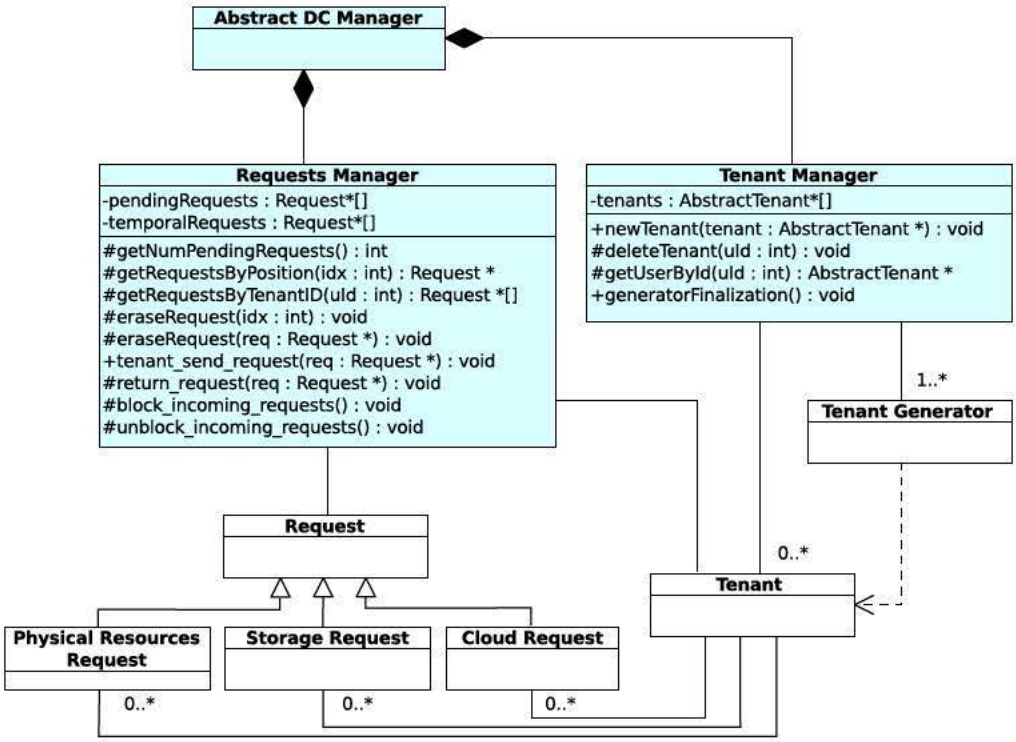


Figure 4.11: Schema of the tenants management model

Data center resources management model

The data center resources management model is based on supplying operations to control storage, features, and usage of underlying architectural resources that build up the data center. Thus, due to the necessity of having physical resources structured to be managed, the *Machines Map* is designed to admit the structure classification of computing and data servers. Therefore, servers are grouped as *NodeSets* inside the *MachinesMap* by the type of node, considering different nodes those built with at least one distinct hardware device. This structure allow model heterogeneous and homogeneous data centers.

Moreover, the data center resources management model are designed sharing its responsibilities in two blocks, allowing to perform decisions based on the underlying architectural features, completing the *Abstract DC Manager* model. Both entities, *Resources Manager* and *Storage Manager*, are linked to the *Machines Map*, as it is presented at Figure 4.12. Only most important operations of each entity have been showed to simplify the scheme.

The *Resources Manager* is designed to acquire data from computing nodes and data storage servers. Major part of operations aimed to get information from a node are designed with a selection parameter (last parameter - *storage : bool*), allowing the selection between computing and storage servers. Moreover, provided operations are classified depending if the characteristic to be obtained is operational or physical. Physical features are extracted from nodes hardware subsystems, e.g., bandwidth of storage devices, speed of processing

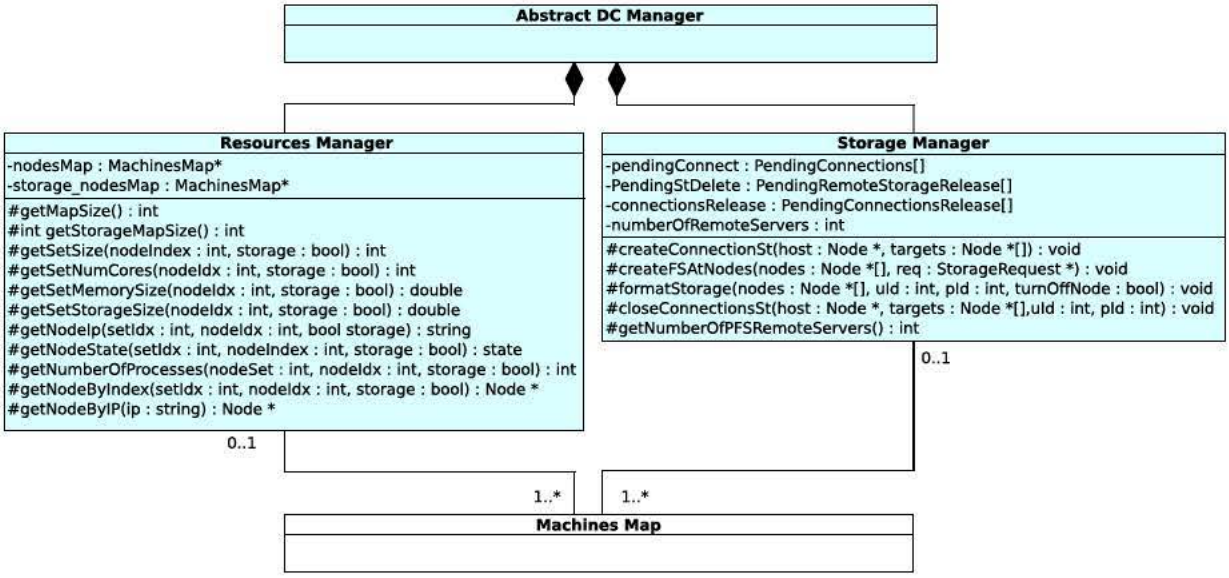


Figure 4.12: Schema of the data center management model

units, number of cores that build the CPUs, amount of RAM memory, IP address. By contrary, the operational features are related to obtain information from applications and VMs, e.g., quantity of VMs allocated into a server, quantity of applications, amount of memory busied, performance status of cores.

The *Storage Manager* design is focused to support operations related to data storage servers. Some of these operations cannot be immediately executed due to the dependency of the network system. To control these processes, the *Storage Manager* has been designed for storing main data from requests into its internal structures, until their completion. Main operations are the creation of connections between node hosts and a set of target storage nodes, the release of storage nodes connections, and the deletion of tenants data when they abandon the system.

Resources provisioning model

The resources provisioning model is the core of the cloud management scheme. It selects the computing servers to host the VMs, and the data servers to allocate tenant data, depending of an user-defined heuristic. Main management responsibilities are spread over following blocks: *Abstract DC Manager*, *Allocation Manager*, *Global Network Manager*, *Cloud Manager*, and *Resources Provisioning Policy*, as it is showed at Figure 4.13.

The management of tenants, requests, and the underlying architecture is responsibility of the *Abstract DC Manager*. As it is described before, it uses the *Requests Manager*, *Tenants Manager*, *Resources Manager*, and *Storage Manager* to orchestrate requests from tenants, communicating the results of the operations to them, and controlling the features of the servers that build up the data center.

The *Allocation Manager* model is responsible for controlling the emplacement purchased virtual resources by tenants as a set of VMs, in order to deploy their applications. It provides structures to manage computing and storage resources each tenants is using,

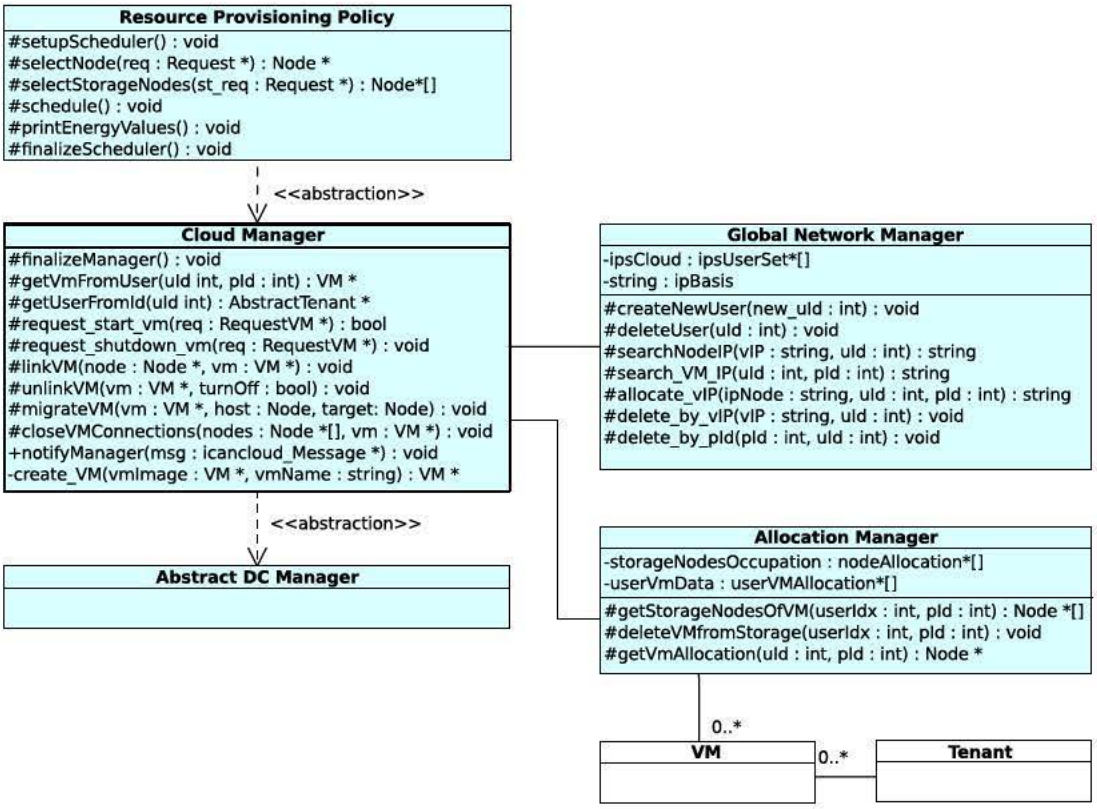


Figure 4.13: Schema of the resources provisioning management model

allowing to release physical resources when a VMs is halted.

The *Global Network Manager*, as it was described at section 4.2.4, has been modeled to manage the IP addresses of cloud computing environments. When a request arrives at network system from a virtual machine, it contains a virtual IP origin and, most of the cases, it is sent towards a virtual IP destination. Due to the fact that L2 and L3 simulation environments do not support virtual addresses translation, the model is designed to manage tables filled with correspondences between virtual IPs (from VMs) and physical IPs (from servers).

The *Cloud Manager* entity has the ability for controlling all resources of cloud computing systems. It inherits all functionalities from *Abstract DC Manager*, allowing to control incoming requests from tenants aimed to obtain resources, and physical resources from computing and data storage servers. In addition, the links to *Global Network Manager* and *Allocation Manager* sets the capability for managing virtual resources. Moreover, Cloud Manager model has its own responsibilities, related to virtual machines management processes.

The *Resources Provisioning Policy* inherits the operations from underlying models to control physical resources, tenants, and virtual machines. Due to this fact, only few operations are needed to be implemented in order to perform different resources provisioning techniques. This model provides to users flexibility for customising existing resource provisioning policies or defining new ones.

4.3.2 Modelling resources provisioning techniques

Nowadays, main goals of resource provisioning techniques in cloud computing environments are focused on providing high performance without harming users SLAs, and reaching an efficient VMs allocation to minimise servers energy consumption. To study the improvement of both, performance and energy efficiency, it is required the analysis of different allocation heuristics. Therefore, the *Cloud Manager* export a set of operations to be completed by the *Resources Provisioning Policy* in order to define the resource allocation technique. They operations are presented at Figure 4.14.

```
1 Node* selectNode (Request* req);
2 vector<Node*> selectStorageNodes (Request* st_req);
3 vector<Node*> remoteShutdown (Request* req);
4 void setupScheduler();
5 void schedule ();
6 void finalizeScheduler();
7 void printEnergyValues();
```

Figure 4.14: Operations to define the resource provisioning policies

Operations to define a resource provisioning technique are classified as: VMs operations, scheduling events, and data saving. First group focus operations related to nodes selection for allocating VMs, remote storage selection, and shutting down a VM. These are described in detail next:

- *selectNode*: The operation for selecting a node allows to define which server is selected to host the virtual resources rented by a tenant. The request managed by this operation contains the type of VM image offered by the IT provider and purchased by him.
- *selectStorageNodes*: A set of nodes from data storage servers may be selected to allocate the resources requested by a tenant. It is possible to return a single data server or a set of data servers where the VM will be able to store data.
- *remoteShutdown*: This method is activated before to shutdown a VM. It has to return the data servers that the VM is using as remote storage, or an empty vector if it only has local storage. It is resources provisioning responsibility for controlling the servers where VMs have data allocated.

Second group sets operations that aims to schedule tenant requests. Therefore they define the start up of the resources provisioning policy, the scheduling of tenant requests, and the actions to be performed before a simulation experiment ends. Those are described following:

- *setupScheduler*: This method allows to initialise internal structures.
- *schedule*: This method must contain the sequence of tasks to be performed for analyse the pending tenant requests. The schedule operation is activated periodically by Cloud Manager in order to select tenants requests to be executed. The time between scheduling events is customizable by user before to launch a simulation experiment.

- *finalizeScheduler*: The operation to finalize the scheduler is activated before to finish a performing simulation experiment. It allows to storage data and release dynamic resources used during the simulation. In addition it is possible to extract results from a simulation experiment after its finalization, exporting data to an external file.

Finally, the third group consists on a single operation to log the energy data measurements. It is described next:

- *printEnergyValues*: In order to record energy consumption variations of a data center, this method is invoked periodically by the Cloud Manager. Print energy values allows to define the data that will be written into a file in order to obtain energy measurements. Moreover it is possible to customise the data recording, and the period between measurements by modifying Cloud Manager features.

The first group sets operations that have direct influence on the physical resources: *selectNode* and *selectStorageNodes*. The operational flow of this operations may involve variations on the energy consumed by the system, the performance of applications, and QoS of provided to tenants. In this scope, two operations are designed to support the initialisation and finalisation processes of the resources provisioning model: *setupScheduler* and *finalizeScheduler*. Otherwise, the operations *schedule*, *remoteShutdown*, and *printEnergyValues* included at second and third group, are independent of the physical resources, focusing the management of user requests (e.g., priorising tenant requests, new models of requests) and different energy data log models.

Next, models for managing tenant requests, generating an energy data log model, and two policies for managing physical resources are presented.

Modelling operations for managing user requests

The operations that exhibit the management of user requests into the resources provisioning model are defined by Algorithms 11 and 12. They present a model for scheduling tenant requests, and releasing data servers when a VM is shutting down.

Algorithm 11 shows a model for analysing tenant requests. First decision is for blocking the incoming messages (see line 3) to avoid new messages during the execution of the operation. Each iteration, the algorithm distinguishes between a storage request (line 5), a cloud request (line 9), or an unknown operation (line 23) which rises an error.

When a cloud request is analysed, the actions performed are divided depending of the type of operation requested. If the operation is for starting a VM, the algorithm launch an operation inherited from the Cloud Manager, *request_ start_ vm* at line 11. Moreover, if the request is for releasing resources (line 12), the operation *request_ shutdown_ vm* release the resources, simplifying the design of the resources policy design. Otherwise, if the incoming operation is activated due to a tenant decides to left the system (to abandon the system at line 16), all its purchased resources are released, erasing the data from internal structures by the *Allocation Manager* (see line 19).

Finally, incoming requests are enabled at line 32. Once this operation is performed, all stored requests received at *Requests Manager* during the recently finished scheduling process, are prepared to be processed the next activation of *schedule* operation.

Algorithm 11 Resources provisioning policy - schedule operation

```
Require: Activation by Cloud Manager
 1:      0
 2:      = getRequestByPosition( )
 3: block_incoming_requests()
 4: while (                                  ) do
 5:    if (isStorageRequest(     )) then
 6:                 getNodeByIndex(     .getNodeSetId(),     .getNodeId(), false)
 7:       AbstractDCManager::userStorageRequest (     ,       )
 8:       eraseRequest(     )
 9:    else if isCloudRequest(     ) then
10:       if (     .getOperation()     REQUEST_START_VM) then
11:          request_start_vm (     )
12:       else if (     .getOperation()     REQUEST_FREE_RESOURCES) then
13:          request_shutdown_vm(     )
14:          eraseRequest(     )
15:          requestErased     true
16:       else if (     .getOperation()     REQUEST_ABANDON_SYSTEM) then
17:                   getUserById(     .getUid())
18:              .deleteAllVMs()
19:          deleteUser(     .getUid())
20:          eraseRequest(     )
21:          requestErased     true
22:       end if
23:    else
24:       Error:Unknown operation.
25:    end if
26:    if (                    ) then
27:             + 1
28:    end if
29:    requestErased     false
30:          getRequestByIndex( )
31: end while
32: unblock_incoming_requests()
```

Next, a model for the *remoteShutdown* operation is presented at Algorithm 12. This operation focuses the release of storage resources associated to a virtual machine. In this model, the storage release operation supports the Cloud Manager operations *request_shutdown_vm* and *deleteUser*. Besides, all resources used by requested virtual machines are released to be purchased by other tenants (see lines 2 and 4). In order to perform Cloud Manager control operations, the operation must return the nodes that have released resources (line 5).

Algorithm 12 Resources provisioning policy - storage release operation

```
Require: A Cloud Request
1: Let [       ] to storage the nodes where a tenant have remote data
2: [       ]    getStorageNodesOfVM(reqVm.getUid(),reqVm.getPid())
3: if ([       ].size()    0) then
4:              deleteVMfromStorageNodes(reqVm.getUid(), req.getPid())
5: end if
   Output:[       ]
```

Generating an energy data log model

The operation described at Algorithm 13 presents one model to extract the energy values from the devices that composes the data center servers. The main strategy is to allocate the energy values into variables. This allow to structure a log file with the data considered necessary by users for their experiments.

Algorithm 13 Resources provisioning policy - Print energy values

```
1: Let                                  getMapSize()
2: Let                                  getStorageMapSize()
3: Let                                  0
4: Let                                  0
5:   Compute nodes
6: for (       to                                ) do
7:                                      getSetSize( , false)
8:    for (       to                               ) do
9:                 getNodeByIndex( , , false)

10:         Get instant power measurements
11:                                             .getCPUInstantConsumption()
12:                                                 .getMemoryInstantConsumption()
13:                                                .getStorageInstantConsumption()
14:                                             .getNICInstantConsumption()
15:                                             .getPSUConsumptionLoss()

16:         Get energy accumulated measurements
17:                                           .getCPUEnergyConsumed()
18:                                               .getMemoryEnergyConsumed()
19:                                              .getStorageEnergyConsumed()
20:                                           .getNICEnergyConsumed()
21:                                           .getPSUEnergyLoss()

22:         Get total measurements of the node
23:                                                .getInstantConsumption()
24:                                            .getEnergyConsumed()

25:         Accumulate data of the total servers
26:                                                                  +
27:                                                              +
28:       PrintToFile (simTime(), ....)
29:    end for
30: end for
31:   Storage Nodes
32: for (       to                              ) do
33:       Analog to the compute nodes
34: end for
35: PrintToFile (simTime(), ....)
```

The operation *PrintEnergyValues* extracts the measurements of energy and power consumed by nodes from the *Machines Map*. As it is shown at algorithm, the compute servers are separated from storage nodes (line 5 and 31), being analog both scheme for getting the consumption values. The Machines Map groups the data center servers as heterogeneous types. Due to this fact, there is a first loop (line 6) that allows to access to each heterogeneous set. Besides, a second loop (line 8) provide the access to each individual

node. Therefore, if this structure changes, the model presented to access to each node in order to print energy values, will need to be adapted.

Once the node is accessed, the instant power measurements for each single device are obtained using internal node operations (from line 11 to 15). The same actions are performed to obtain the energy data accumulated from the beginning of a simulation experiment (from line 17 to 21). All these data are provided by the energy managers that composes on the energy model of the nodes.

Besides, the total power consumption and the energy consumed are extracted from the node (line 23 and 24). When all required data are stored at variables, it is possible to print the information to a file. Lines 28 and 35 mask the print format process of the data. Users may format data to be imported by different software applications, e.g., csv, txt, gnu.

The operation *PrintEnergyValues* is activated periodically. The interval between activations may be configured by users at Cloud Manager parameters.

Modelling first fit resources provisioning model

A resources provisioning model is a bin packing problem where VMs with different requirements must be packed into a finite numbers of servers (bin containers), minimizing the number of servers used. This is in terms of computational complexity theory a NP-hard problem.

A provisioning policy can be designed much more effective than first fit resources provisioning model. Despite the fact that this model does not guarantee an optimal solution, it is a fast model to obtain a solution, allowing to adjust the energy consumed due to each VM is placed into the first server that the algorithm selects, until each server is not able to allocate more VMs. Therefore, when a server can not allocate a VM, the next node capable to provide the purchased requirements is selected.

Following the first fit resources provisioning algorithm is modelled using the iCanCloud APIs, as it is presented at Algorithm 14.

The algorithm contains two loops (lines 10 and 17) to get a solution. Due to the fact that servers are ordered as homogeneous sets into the machines map structure, first loop go through these sets to obtain the features of the set of nodes. Main features searched are the total physical memory and the amount of computing resources that built in the servers. Using these data, first fit algorithm can calculate if the type of servers selected have the capability to allocate a VM, concretely the amount of resources (virtual memory and computing power) requested by the tenant (line 11. Afterwards, second loop is responsible for search the first node allocated at structure that is able to host the new VM.

The condition to select a node that has enough resources to allocate the purchased resources by tenant, is the quantity of VMs running into the system. The VMs allocated into the chosen server may not exceeded the *maximum_number_of_processes_per_node* parameter (line 22). This parameter may be customized by users at Cloud Manager properties. If the number of VMs allowed into a server is too high, it may harm the performance of applications running into it. Therefore, if the applications executing in a node reduce considerably its performance, the cloud provider may violate the SLA agreement purchased by tenants.

Algorithm 14 First fit resources provisioning model

```
Require: A cloud request with the requisites of the VM to be located
 1:   Push in the set the different heterogeneous node types where it is possible allocate the vm
 2: Let                                  getMapSize()
 3: Let                              .getSingleRequestType()
 4: Let                                .getMemorySize()
 5: Let                               .getNumCores()
 6: Let      to allocate selected type of nodes that will be selected as result
 7: Let        to allocate selected node as result of the provisioning policy
 8: Let                _       _   _         _    _        to define the maximum number of VMs permitted
    in a Node

 9:   Get the server sets
10: for (       to                          ) do
11:    if ( (getMemorySize( , false)                )    (getNumCores( ,false)              ) ) then
12:          .push_back(i);
13:    end if
14: end for

15:   Select the first set
16:             false
17: for (       to    .size(),            ) do
18:                 getSetSize(    [i],false)
19:    for (       to          ,          ) do
20:                 getNodeByIndex(    [i],j)
21:                                .getNumOfLinkedVMs()
22:       if (                                 _       _   _         _    _      ) then
23:                      true
24:       end if
25:    end for
26: end for
    Output:
```

Once the node is found, the the first fit algorithm finish. Therefore, in order to break the loops flow, the variable *found* is set as true (line 23).

Modelling best fit resources provisioning model

The best fit resources provisioning model is a policy designed to locate tenants purchased resources (VMs) more effective than first fit. The computational complexity of best fit is the same than worst case of first fit algorithm ($O(n)$). However, the best fit algorithm preserves the performance of applications, obtaining a better emplacement for the VMs.

The aim of best fit algorithm is to find the server with the minor number of VMs running, distinct to zero, in order to place a new one. The algorithm behavior for a low number of VMs is similar to the first fit. Moreover, best fit algorithm improves first fit when tenants decide to finalise VMs, being its resources released from the system. This fact produce free resources into unknown servers to best fit. By contrary to fist fit policy where first server with enough resources to allocate a new VM is selected, best fit policy analyses all servers to allocate new VMs, allowing to maintain the same amount of purchased resources into each node. This fact, improves the performance of the applications, minimizing the energy consumption of the servers.

The model for best fit provisioning scheduler is presented at Algorithm 15.

Algorithm 15 Best fit resources provisioning model

```
Require: A cloud request with the requisites of the VM to be located
 1:   Push in the set the different heterogeneous node types where it is possible allocate the vm
 2: Let                              getMapSize()
 3: Let                          .getSingleRequestType()
 4: Let                             .getMemorySize()
 5: Let                           .getNumCores()
 6: Let      to allocate selected type of nodes that will be selected as result
 7: Let            to allocate the temporal node selection
 8: Let           to allocate the selected node as result of the provisioning policy
 9: Let              _        _   _        _   _        to define the upper amount of VMs in a Node

10:   Get the server sets
11: for (       to                    ) do
12:    if ( (getMemorySize( , false)          )   (getNumCores( ,false)          ) ) then
13:          .push_back(i);
14:    end if
15: end for

16:   Select the best set
17:              NULL
18: for (       to     .size()) do
19:                 getSetSize(    [i], false)
20:    for (       to          ) do
21:                      getNodeByIndex(    [i],j)
22:                                 .getNumOfLinkedVMs()
23:       if (                              _        _   _        _   _      ) then
24:          if (             NULL) then
25:
26:          else if (          .getState()    OFF) then
27:
28:          else if (                    0) then
29:             if (          .getNumOfLinkedVMs()                ) then
30:
31:             end if
32:          end if
33:       end if
34:    end for
35: end for
    Output:
```

First step of the algorithm choose the servers sets where the VM may fit (line 11). The condition for selecting the type of node is the quantity of cores and the amount of memory that built-in the server type (line 24). Once the servers set are picked up into the structure, the policy is prepared for search the concrete server to allocate the resources requested by tenant.

The second part of the algorithm, analyses all servers (lines 18 and 20) saving the node that fits better with requested resources as (line 17). Firstly it is necessary to know if the server has reached its upper limit to allocate VMs. If it may serve enough virtual resources and has not been initialized yet, this server is the best choice (line 24). Moreover, once has been initialised, it is important to distinguish between on state and off state nodes. Therefore, the second condition (line 26) has been designed to select a

node that will not be full and neither at off state. This case may be possible when
is initialised with an off server, avoiding the possibility to select as result a node with less processes running than the others due to the node is off. Finally, the third condition is based on analyse if the number of processes of the node is minor than the processes of best node selected until that moment (line 29).

4.4 Summary

This chapter presents the simulation models for virtualising cloud computing environments. The design of this model exhibits the hypervisor model, the management of tenant requests, network addresses management, and physical resources hostage.

The new contributions presented in this chapter are the models of the virtualization layer of a cloud computing environment. This contributions are grouped in both: the hypervisor model, and resources provisioning model. The hypervisor model has been designed providing flexibility to users, in order to improving existent hypervisor scheduling policies of each hardware subsystem. In addition, real scheduling policies for managing CPU and Memory resources were modeled using the hypervisor's APIs of the iCanCloud simulation platform. The resources provisionig model, has been presented jointly to the Cloud Manager model. They manage the amount of resources of the the servers that build up the models of data centers. These are be exhibited in detail, showing all functional blocks with main APIs to manage them.

The contents of this chapter complete the entire model of the iCanCloud simulation platform, having all the necessary parts to simulate cloud computing systems. After the simulation platform has been designed, modeled, and developed, it is necessary to test all systems for validating its correctness.

Chapter 5

Evaluation

This chapter presents a set of experiments targeted to achieve two different objectives. Initially, several experiments have been conducted in order to check the accuracy of the proposed simulation platform. The main objective of these experiments is to show the level of accuracy obtained from simulation, by comparing the results obtained from the real scenario with the results obtained from iCanCloud.

Next, several performance and energy consumption experiments have been performed. These experiments aim to present the main capabilities of iCanCloud. In order to validate the models of iCanCloud, the simulation platform has been evaluated by modelling real applications, and performing simulations-real executions-mathematical model results comparisons. Besides, the energy models has been validated by comparing simulation results and real traces obtained from mechanized nodes.

The level of flexibility of iCanCloud has been tested by executing different workloads in different data centre scenarios, analysing both the overall system performance and its energetic consumption, in order to exhibit the ability to foresee the behaviour of a real system.

Finally scalability experiments have been performed to test the memory and computing requirements of iCanCloud platform when large simulation models are executed.

5.1 Evaluation of the iCanCloud simulator

After a simulator has been developed, implemented, and debugged, it must be tested for correctness and accuracy. However, determining that a simulator is absolutely valid over the complete domain of its whole intended field of applicability is a very hard and time-consuming task. Thus, the level of accuracy of a given simulator can't be calculated for the entire domain this simulator is targeted using a single value, because this accuracy depends directly of the system to be modeled.

Validating a model means "substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model" [143].

In this section a validation process has been conducted to demonstrate the applicability and usefulness of the iCanCloud simulator. This process consists of two parts. First

part compares the results obtained from a validated mathematical model of the Phobos application [144], with results from the analogous model using iCanCloud. This model consists of the simulation of the application, and the corresponding hardware environment where that application has been executed. Second part compares the results obtained from executing the Phobos application in the Amazon EC2 system, with those results obtained from simulating the same executions using iCanCloud.

The application chosen for this validation calculates the trajectories of Phobos, the Martian moon, in the context of the Finnish-Russian-Spanish Mission to Mars that was launched in 2011 [145], which was ported to the Amazon EC2 public cloud infrastructure [144]. Pertaining to the parameter sweep execution profile, the resulting application divides the overall tracing interval in subintervals that are calculated by the subsequent tasks in the cloud – thus the tracing interval of a task is not related to its execution time. The tracing interval processed by each task is the same and the system performs dynamic scheduling where a continuous polling of free cores guarantees a constant resource use.

As was explained before, the chosen cloud infrastructure is Amazon EC2, which has become the de facto standard public cloud infrastructure for many scientific applications. The baremetal infrastructure providing the services is located in two locations in USA, one in Asia and another one in Europe. The users may choose from a wide range of machine images that can be booted in one of the offered instance types. Depending the chosen instance type, the number of CPU core number, core speed, memory and architecture differ, as shown in Table 5.1. The speed per CPU core is measured in EC2 Compute Units, being each C.U. equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Nevertheless, accessing to an almost infinite computing infrastructure has its price, which depends on the instantiated VM type per hour.

The paper which describes the porting of the Phobos tracing application [144] introduced and validated an execution model, along with a study of the best infrastructure setup by means of instance types and number. In order to deal with the complexity level added by an infrastructure following a pay-as-you-go basis, a metric named Cost per Performance ( ) was also provided:

——— ——— (5.1)

where is the task execution time, the values of and correspond to the whole tracing interval and the tracing interval per task, that is, the grain of the application. On the other hand, and are the number of Virtual Machines and number of cores per Virtual Machine, as shown in Table 5.1 along with the machine's usage price per hour ( ). This way, the best infrastructure setup would be that which produced the lowest C/P value.

Figures 5.1, 5.2, 5.3, 5.4, and 5.5 present results obtained from executing the model of Phobos application along with the results of the same application implemented on iCanCloud. Each figure represents the C/P metric for the experiments, where the small, large, extra large, high CPU medium and high CPU extra large instance types provided by Amazon are used, and number of VMs and tracing intervals are varied.

Mainly, the most relevant difference between the iCanCloud and the mathematical model is the variations obtained when the number of VMs increases. In those results obtained using iCanCloud, we can see that in some cases, using the same size for the interval (in years) and increasing the number of VMs, causes an increase in the C/P

Table 5.1: Characteristics of the different machine types offered by Amazon EC2. C.U. corresponds to EC2 Compute Units per core, the equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

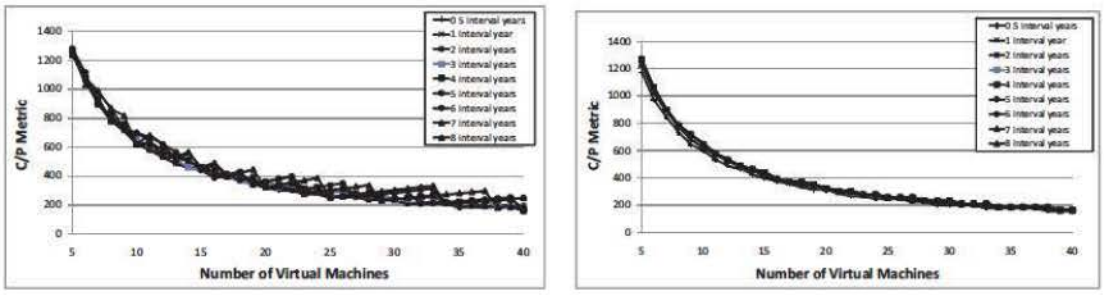
Machine Type	Cores	C.U.	Memory	Platform
Standard On-Demand Instances				
Small (Default)	1	1	1.7GB	32bit
Large	2	2	7.5GB	64bit
Extra Large	4	2	15GB	64bit
High CPU On-Demand Instances				
Medium	2	2.5	1.7GB	32bit
Extra Large	8	2.5	7GB	64bit

metric, which does not happen in experiments using the mathematical model of Amazon. It is mainly caused because the cost of the each VM is measured in completed hours, whereof an hour cannot be split in fractions. Then, increasing the number of VMs provides the same execution time, which produces a increasing of the cost for this configuration. Logically, the greater number of VMs used, the greater cost of the system. This effect only appears when the number of VMs gets higher. When the number of VMs is low, the performance gain when more VMs are used justifies the increase in the cost. Moreover, the mathematical model does not represent the time spent on performing I/O operations, instead iCanCloud.

The best results are obtained in those tests that use small and high CPU medium instances of VMs (see figures 5.1 and 5.5). In those cases, the cost of increasing the number of VMs for reducing the execution time and then using less complete hours for the execution is similar to the cost of using less VMs that requires more complete hours to execute the testbed. It can be shown by comparing the simulations and the model, whereof both result shows practically the same shape.

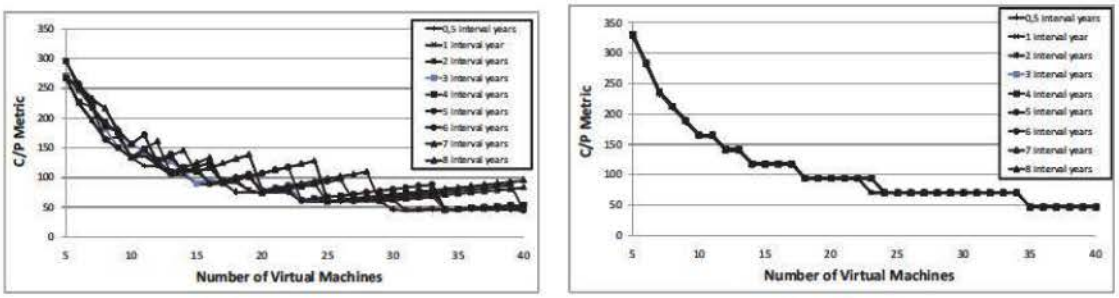
Otherwise, figures 5.2, 5.3 and 5.5 show a more noticeable difference between the simulation and the model. This difference can be appreciated in those charts is because the C/P metric values are low, and then the differences between the simulation and the model are shown as peaks. Those peaks basically represent the maximum value of C/P, which means that the time required for executing this testbed consumes a little portion of the last hour of the total execution time. Thus, it means that using more VMs the testbed can be executed without requiring that portion of hour, and then there is no need of paying for an entire hour, obtaining a considerable drop in the C/P value.

We can conclude that both iCanCloud simulations and the mathematical model show the same tendency in the trade-offs between cost and performance. However, simulations show a more realistic behavior due to more details of the system are modeled. The main advantage of simulations versus the mathematical model is accuracy, and the main drawback is obviously the execution time. iCanCloud requires more physical resources to be executed in a computer due to the detailed model of simulations, than mathematical model to be performed.



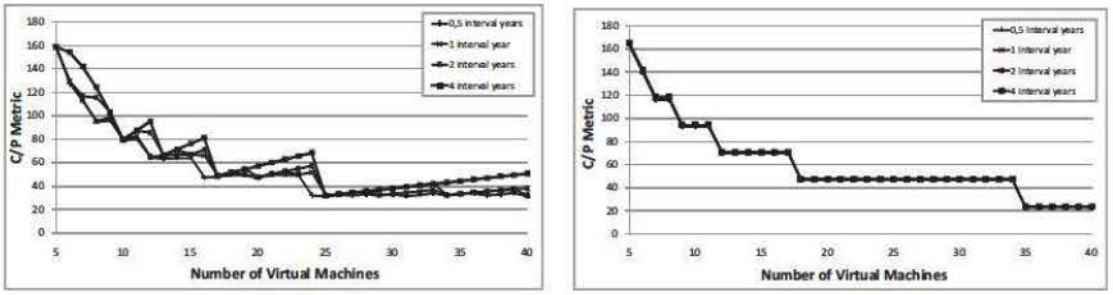
(a) iCanCloud (b) Model

Figure 5.1: Simulation versus mathematical model of Phobos using small instances



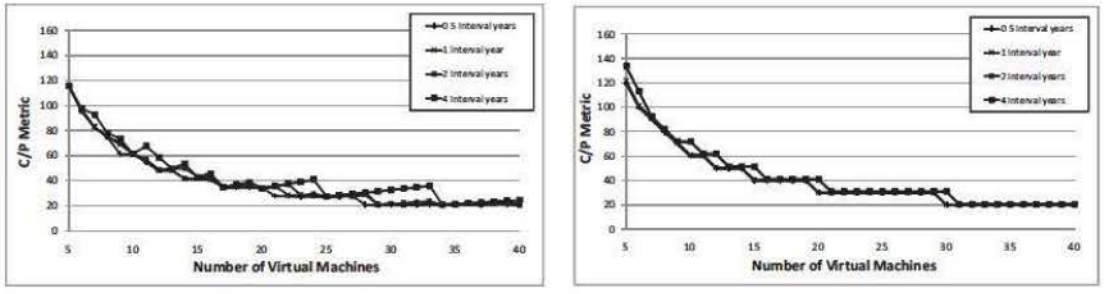
(a) iCanCloud (b) Model

Figure 5.2: Simulation versus mathematical model of Phobos using large instances



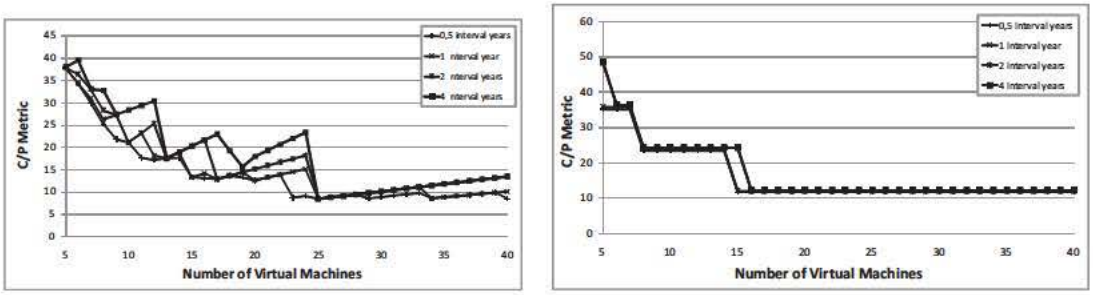
(a) iCanCloud (b) Model

Figure 5.3: Simulation versus mathematical model of Phobos using X-Large instances



(a) iCanCloud (b) Model

Figure 5.4: Simulation versus mathematical model of Phobos using high CPU medium instances



(a) iCanCloud (b) Model

Figure 5.5: Simulation versus mathematical model of Phobos using high CPU X-Large instances

Figure 5.6 shows a dispersion chart that compares the results of executing the Phobos application in the Amazon EC2 system, against the analogous model using iCanCloud. In those tests both different interval years and a different number of VMs have been used. Results obtained from those tests executed in Amazon EC2 are represented using hollow forms. Otherwise, iCanCloud results are shown using filled forms.

Initially, Phobos application has been launched using 0,5 interval years (represented as squares). This chart shows that both the results executed in the real environment and the results obtained using iCanCloud follows the same tendency. However, those experiments that obtains lower C/P values fits better with the simulation.

Those experiments that use 1 interval year are represented as triangles. In those cases, the C/P value obtained is greater than using a 2 interval year. This is due to the behavior of the Phobos application with different interval years. This asymmetry was already observed during its first porting onto the cloud [144].

In general, in those tests that obtain the lower values of the C/P metric, iCanCloud obtains practically the same results. Otherwise, when the C/P value increases, there is a slightly difference between the real system and the simulation. Finally, all results obtained both from the Amazon EC2 and iCanCloud follows a saw-tooth waveform, which can be also appreciated in the previous charts shown in figures 5.1, 5.2, 5.3, 5.4, and 5.5.

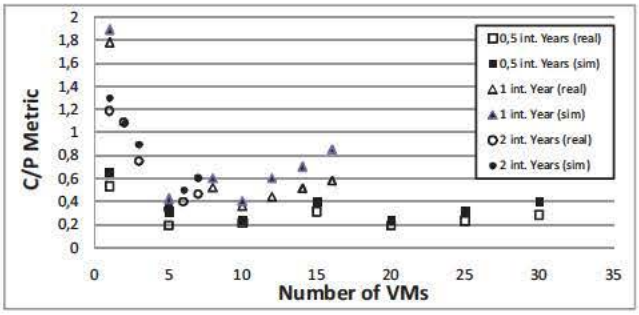


Figure 5.6: Simulation versus Amazon EC2 execution of Phobos using small instances

5.2 Validation of the energy model of iCanCloud simulation platform

After the simulation framework has been evaluated, this section exhibits the validation of the energy model of iCanCloud simulation platform. The process for validating the energy model involves generating test cases, simulating those tests, and comparing simulation results to a known reference. In order to obtain the accuracy of the power usage, we present a methodology which we have followed for validating the scope of the energy models of the proposed framework, and how simulation results match tests run on bare metal.

5.2.1 Bare metal power measurement methodology

In order to run tests on bare metal and compare results to simulated results, a method for measuring power consumption is required. There are several ways of measuring the power consumption of a single system. These approaches are described next, ordered from less intrusive methods to more intrusive methods.

The first method for estimating power usage consists in using mathematical power models which describe how a system or its components consume power based on resource utilization. It has the advantage of being totally unobtrusive, and scaling to an arbitrary number of nodes is easy. However, power models do not provide the precision of hardware devices, and parameters can vary across different hardware devices [146].

The second method consists in using a power meter which measures the power consumption of the whole system. One popular device is Watts Up Pro [147]. This device is essentially a power outlet in which a system can be plugged in, and provides several interfaces for reading power consumption in real time, typically with a frequency of 1-4Hz. These devices are ideal for measuring the aggregated power consumption of a node, but do not provide power consumption for individual components. In fact, estimating the exact amount of power consumed by a single component is not only difficult due to the fact that only the aggregated power is measured, but the inefficiency of the power supply unit (PSU) needs to be taken into account. This inefficiency is attributable to the AC to DC conversion which is performed by the PSU. While a hardware device is required for measuring power consumption, making this method impractical for medium and large-scale systems, no hardware or software modifications are required for the measured system.

A third approach for measuring power consumption consists in instrumenting a system with multimeters for measuring voltage and currents of the individual power supply cables, which connect hard disks and motherboard with the power supply unit (PSU). In contrast with the previous approach, the energy which is lost in AC to DC conversion in the PSU is not measured, and the currents which flow through individual cables can be measured independently and matched to the power consumption of individual components. In addition, the sampling rate is much higher, meaning high precision can be achieved. The drawback of this approach is the fact that system instrumentation is highly intrusive, expensive to scale, and impractical for more than a few systems.

In this work we have chosen to combine the second and the third method for validating our framework, namely power meters and multimeters, which we use for obtaining precise measurements of the test cases described in Section 5.2.3.

5.2.2 Hardware setup

In order to obtain detailed power measurements, we have instrumented several systems with power meters and multimeters which are directly attached to the motherboard. Our setup is depicted in figure 5.7 and described in more detail next.

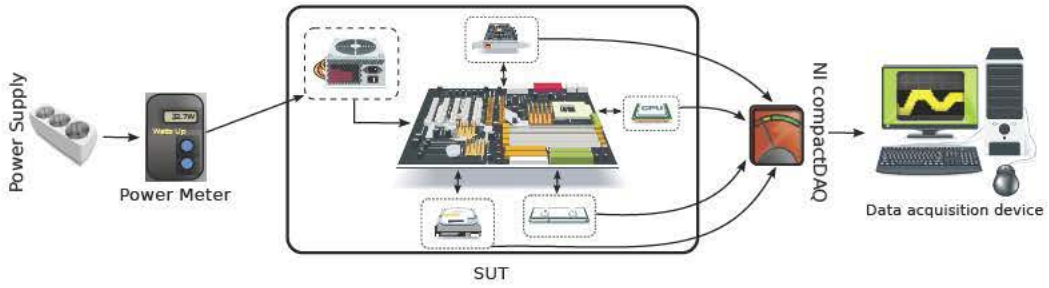


Figure 5.7: Hardware set-up for running the validation experiments

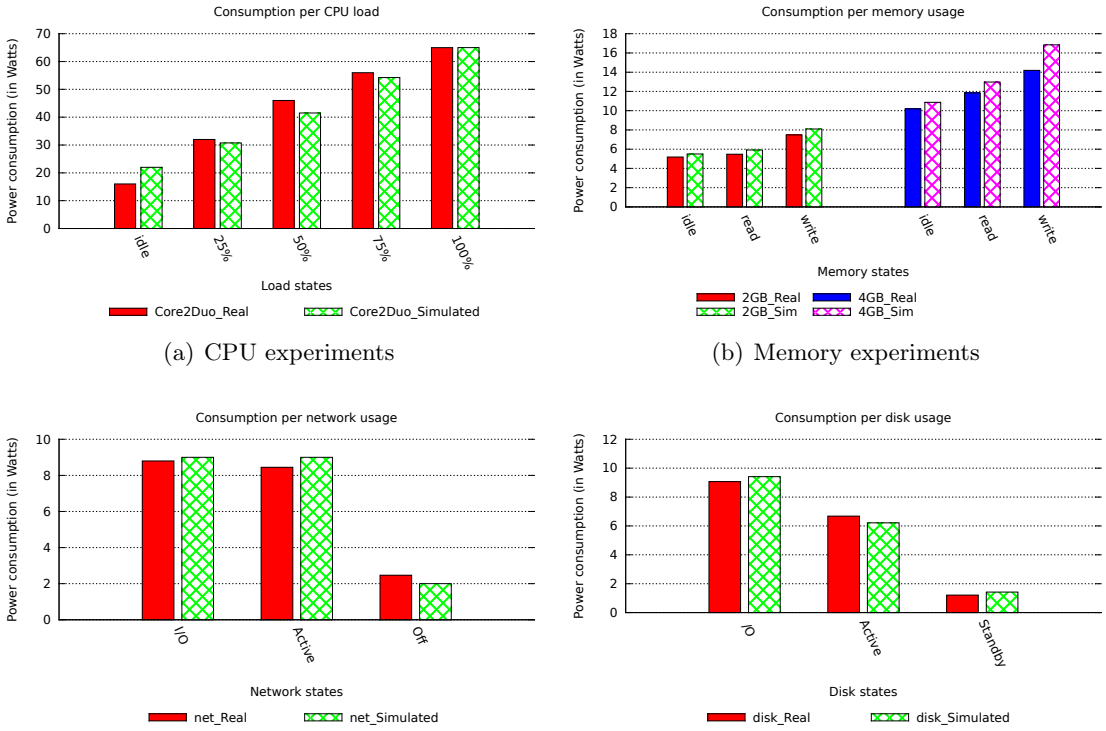
For each experiment, we used a system which is instrumented and used to run test cases and benchmarks, and another system which collects the data obtained from our power measurement devices. This second system is required in order to eliminate potential measurement overheads when storing and processing the acquired power measurement data. Our power measurement devices consist of a Watts Up! Pro power meter, and a National Instruments cDAQ-9174 which is attached to the power connectors of the motherboard. These devices are connected to a separate system which is only used for collecting, storing and processing the data using NI LabView.

5.2.3 Power measurement test cases

In order to measure power usage of individual components, we developed a series of independent test cases which stressed a single component. These experiments target CPU, main memory, network, and hard disk. In order to measure the CPU power consumption, a program which runs an infinite loop achieves peak level utilization of a single core. By spawning multiple threads, more than one core can be put at peak level utilization. Moreover, a fraction of peak level utilization can be achieved by interleaving computation with increasingly large intervals of CPU idle time. As a result, it is possible to set CPU utilization to 0%, 25%, 50%, etc. and up to 100%. We ran this benchmark, varying CPU utilization from idle to peak load, while acquiring data from our NI device at a sampling rate of 5 Khz (every 200 μs). The same benchmark was modelled and ran in the simulator. The comparative result is depicted in Figure 5.8(a).

The memory, however, is more complex, since we observed that the operating system and other programs running in the background impacted its power usage significantly. For that reason, we ran no operating system and booted memtest directly. We ran memory test 1, which writes and then reads every address. The same memory accesses were performed in the simulator, and the resulting power consumption estimation is depicted in Figure 5.8(b).

In order to test the network, we ran three different tests. First, we left the network card off by unplugging the network cable. Second, we plugged it in but ran no communication



(a) CPU experiments

(b) Memory experiments

(c) Network experiments

(d) Disk experiments

Figure 5.8: Validation experiments to measure individual components

tests (idle state). Third, we ran iperf, a network bandwidth benchmark. Figure 5.8(c) demonstrates several power consumption states.

For testing the disk, we ran sequential reads/writes, varying block size and total I/O size. Figure 5.8(d) shows the results of the validation process for I/O experiments.

The analysis of the results obtained is performed using estimators of the accuracy and the comparison trend. The main estimator is the average accuracy of each test with different parameters. Another significant method to estimate the results is to check the comparison trend of those tests. This comparison trend is the one that results from comparing the real and the simulated results of one test when certain parameters are varied. Each parameter value incorporates a pair of real/simulated values that is treated as a point into the comparison trend line. The more the trends line mimics the chart of the identity function graph (Y = X), the more accurate the simulation is. In this validation process the Pearson coefficient has been used. Basically, this coefficient measures the level of linear dependency between real and simulated results, which indicates the straightness of the trends line. The nearer this coefficient is to 1, the better the correlation between the real and the simulated results.

The results of the average accuracy values are collected in Table 5.2. This table shows that in most cases the accuracy of the simulator is greater than 85%. The cases where we obtain a poor accuracy is when the hardware components are in its idle (or standby) states. This is caused because our model uses the energetic values obtained from the

System	Device state	Accuracy (in %)	Pearson Coefficient
CPU	Idle	65.19%	0.9874
	25%	95.23%	
	50%	89.09%	
	75%	95.56%	
	100%	99.9%	
Memory	Idle	85.83%	0.981
	Read	91.82%	
	Write	91.91%	
Network	I/O	97.72%	0.991
	Active	93.44%	
	Off	81.01%	
Disk	I/O	75.3%	0.9635
	Active	85.11%	
	Stand-by	85.28%	

Table 5.2: Accuracy estimators of the validation process

device's data sheet to calculate the energy consumed by such device. Generally, these values are lower than consumption obtained in practice. Moreover, due to these values are low, small variations causes greater percentages in the accuracy. However, in the tests that require a variation in the energy consumed by the device, simulation results show a good approximation to the real consumption. Finally, all the tests executed show the same tendency both in the real system and in the simulated environment (see Pearson coefficient in Table 5.2).

Next, a set of performance experiments were executed to analyse the aggregated energy consumption of a computing node. In order to perform this task, the BIPS3D application has been executed using a different number of processes. The BIPS3D application has been modelled using SIMCAN in the past, where the generated models were used to study the performance of BIPS3D in distributed systems using different architectural configurations [138]. The energy consumed of these experiments has been measured using the Watts Up! Pro power meter in three different computing nodes. The main features of each computing node are described below:

- Computing node with a Core2-Duo CPU, 4 GB of RAM memory and a Gigabit Ethernet network.
- Computing node with a Quad-Core CPU, 2 GB of RAM memory and a Gigabit Ethernet network.
- Computing node with a AMD Opteron 12-Core CPU, 64 GB of RAM memory and a Gigabit Ethernet network.

Figure 5.9 shows a comparison of the energy consumption between the execution of BIPS3D in both real and simulated environments. Each experiment was executed using 2, 4, 8 and 16 processes. The main objective of this experiment is to measure the tendency of energy consumption when different hardware configurations are used. This chart shows that increasing the number of processes has an insignificant impact on the overall energy

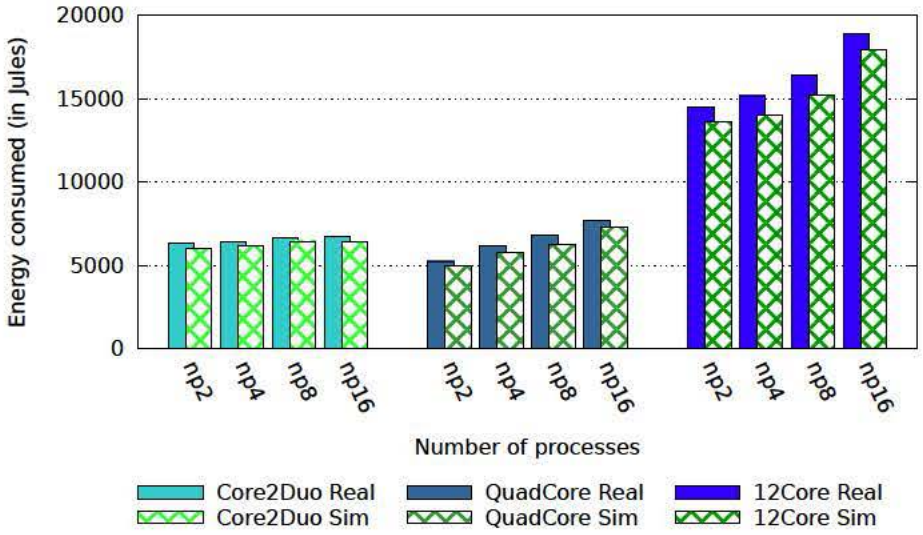


Figure 5.9: Real vs. Simulation - Energy consumption of BIPS3D

consumption in the Core2-Duo CPU. On the contrary, in the Quad-Core and 12-Core CPU, increasing the number of processes raises the energy consumed as well. This is mainly caused by the idle cores and the p-states of the CPU. The execution in the 12-cores CPU shows a considerable energy consumption, being in some cases four times greater than the other CPUs. However, simulation results reflect the same conclusions than results obtained from the real executions. In fact, the tendency of the obtained results when different hardware configurations are used is the same.

5.3 Performance and energy consumption experiments

In this section, a set of experiments have been executed with the purpose of demonstrating the feasibility of the framework to calculate the estimated energy consumption in different simulated cloud environments. These experiments have been performed using iCanCloud simulation platform. In order to perform this task, different cloud environments and applications have been modelled. Section 5.3.1 provides a description of the cloud environments used in these experiments. Section 5.3.2 contains a description of the applications executed in each modelled environment. Finally, Section 5.3.3 shows the evaluation results and an analysis of the obtained results.

5.3.1 Description of cloud environment models

The experiments described in this section have been performed using two different cloud environments. A detailed description of each environment is provided in Table 5.3.

First, a data centre has been modelled. Usually, data centres are designed for continuous use, where a large set of Virtual Machines are hosted by executing servers. The data

Datacenter configuration	
Total number of computing nodes	1024
CPU cores per node	12
CPU core speed	20000 MIPS
Disks per node	1
Disk model	ST1000DM003 - 1TB
Memory per node	64GB DDR3
Rated output power of PSU	1000W
Network	Ethernet 1 Gbps / 10 Gbps
Scientific Cloud configuration	
Total number of computing nodes	128
CPU cores per node	2 and 4
CPU core speed	12000, 14000 MIPS
Disks per node	1
Disk model	Western Digital WD2500JB - 250 GB
Memory per node	2GB DDR2 and 4GB DDR3
Rated output power of PSU	350W
Network	Ethernet 1 Gbps

Table 5.3: Configuration of experiments using different cloud environments

centre used in these experiments consists of 1024 computing nodes, where each node has a 12-Core CPU with 64 GB of RAM memory. The virtual machines deployed consists of 1 CPU, 2GB of RAM memory, and 1GB of storage.

Second, a scientific cloud has been modelled. This kind of cloud environment is mostly used to execute scientific applications, generally in universities and research labs. This scientific cloud consists of 128 nodes, which are divided into 64 nodes with a QuadCore CPU and 2 GB of RAM memory, and others 64 nodes with a Core2Duo CPU and 4 GB of RAM memory.

5.3.2 Description of application models

This section provides a description of the applications executed in the previously described environments. For those experiments performed on the data centre, a web server has been modelled. A web server application is a client-server system. Typically, the web server supports thousands of clients requesting services at the same time. Therefore, the web server, handles requests by different users simultaneously. These requests have to be processed or serviced before another message can be attended. In order to manage the simultaneous requests, the web server enqueue the petitions to process them in order of arrival. Generally, a web server supports an upper bound on the number of hits per day that can be service. In this experiment we use an interval of [9000-10000] requests per hour. As an example, Web servers as wikipedia supports a number of hits per hour nearly to 10,000 [148].

For those experiments executed in the scientific cloud environment, the BIPS3D application has been used [138].

5.3.3 Evaluation

We studied the energy consumption of two different cloud environments, a data centre and a scientific cloud, using iCanCloud. In each cloud environment, different applications have been modelled to study the energetic impact. First, web servers has been modelled to be executed in the data centre. Second, a High Performance Application, called BIPS3D, has been executed in the scientific cloud environment. The experiment focuses the analysis of the energy consumption variability by simulating different physical resources management strategies. To study the entire scope of usage variations of the physical resources, several workloads are modelled to require a specific percentage of resources: 20%, 40%, 60%, 80% and 100%. Moreover, two different strategies to manage the servers of the data centre has been studied in order to adjust the energy consumed by the entire system. First, strategy consists on maintain the resources that are not used in idle state. This is, the resources are connected but no Virtual Machine are executed on them. It allow to use faster the resources when they are needed but, it produces a major energy consumed. Second, the unused resources have been in stand-by. This is, the resources are connected but suspended and waiting to be activated. This mode consumes less energy than idle state, but requires more time when a server is selected to host a VM performing the transition to idle state.

Power consumption results of the data centre environment					
% of workload	State of unused nodes	Max (kW)	Min (kW)	Average (kWh)	Std. deviation
20%	Idle	93.669	73.764	90.064	7.134
	Stand-by	60.896	40.963	57.223	7.181
40%	Idle	144.481	92.652	115.578	9.989
	Stand-by	96.094	68.073	90.961	10.101
60%	Idle	136.994	106.184	131.479	10.987
	Stand-by	120.594	89.784	114.949	11.114
80%	Idle	151.573	118.734	145.695	11.711
	Stand-by	143.333	110.494	137.317	11.846
100%	Idle	165.682	131.891	157.694	4.814
	Stand-by	163.724	131.11	158.384	4.487
Power consumption results of the scientific cloud environment					
% of workload	State of unused nodes	Max (kW)	Min (kW)	Average (kWh)	Std. deviation
20%	Idle	12.965	12.015	12.198	0.352
	Stand-by	4.934	1.874	4.164	0.361
40%	Idle	14.338	12.015	12.721	0.084
	Stand-by	9.329	3.394	7.876	0.682
60%	Idle	14.708	12.015	12.534	0.998
	Stand-by	13.584	4.914	11.403	1.022
80%	Idle	14.919	12.015	12.578	1.164
	Stand-by	14.213	5.434	12.606	1.153
100%	Idle	15.119	12.015	12.614	1.151
	Stand-by	15.119	6.434	12.753	1.176

Table 5.4: Energy consumption results

Table 5.4 shows the results obtained for both environments. This table shows the minimum, maximum, average and standard deviation values of power and energy consumption for each experiment, extracted using the samples obtained from the simulation of one entire day of real execution (24 hours) at a rate of 1Hz.

These results show a significant difference of energy consumption when unused resources are in stand-by instead of in idle state. The main reason of this difference in the energy consumption is the that the components in idle state consume more energy that components in stand-by. However, this difference is practically insignificant for those workloads of 100%, where all the resources of the cloud are active. In these cases, there is no difference for maintaining in stand-by or idle the unused resources.

When the percentage of active resources increases, the energy consumption increases as well. However, this difference does not grow proportionally. The main reason of this is the energy consumed by the unused resources, which have more impact on the overall system consumption when less active resources are used. The standard deviation obtained for each workload, when different strategies are used (idle and stand-by) is practically the same, which reflects that the behaviour of active resources are identical for both strategies.

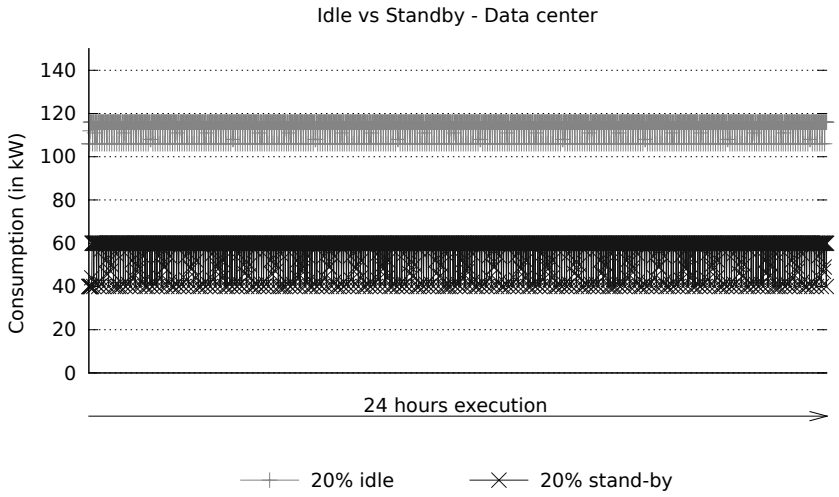


Figure 5.10: Energy consumption of data centre using 20% of resources

In general, the data centre maintaining 100% of resources active, consumes an average of 157.694 kWh when unused resources are in idle state, and 158.384 kWh when unused resources are in stand-by. Note that the difference of consumption is practically insignificant. On the contrary, maintaining 20% of resources active, the energy consumption when unused resources are idle is 90.064 kWh, versus 57.223 kWh when unused resources are in stand-by. In this case, using different strategies produces a difference of energy worthy of consideration. The same occurs in the scientific cloud environment. In this case, the energy consumption by maintaining 20% of resources active when unused resources are in idle and stand-by state is 12.198 kWh and 4.164 kWh respectively. On the contrary, the consumption when 100% of resources are active is practically the same for idle and stand-by strategies, being 12.614 kWh and 12.753 kWh respectively.

To properly dimension the power supply system of the cloud, at least the maximum power estimated must be provided to avoid system power failures.

Due to the fact that resources are rarely operating under peak load, different resource provisioning strategies can have a great impact on energy consumption. We performed a series of experiments simulating different scenarios, which show a great difference in

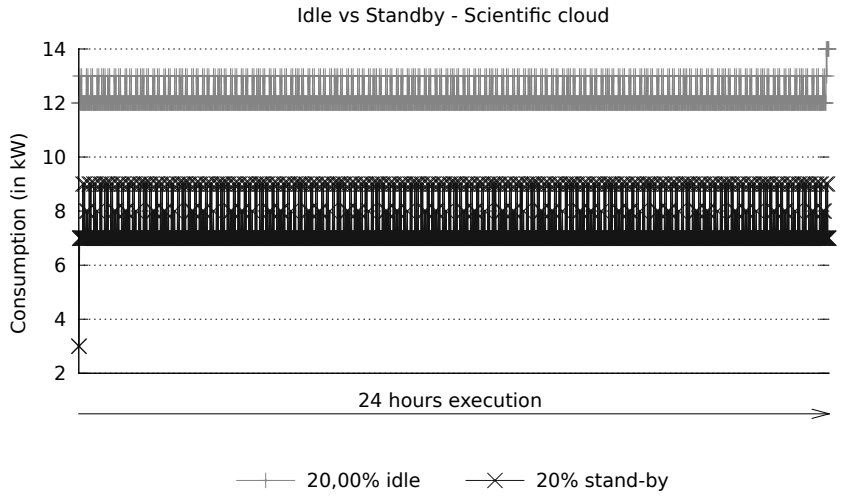


Figure 5.11: Energy consumption of Scientific Cloud using 20% of resources

energy consumption depending on the selected strategy. Figure 5.10 and Figure 5.11 show the average of energy consumption during 24 hours for the Datacenter and Scientific Cloud respectively. In these charts, 20% of resources were active, while the rest of resources could be in idle or stand-by. These figures depict the difference in power consumption using different strategies in each cloud environment.

5.4 Measuring the scalability of iCanCloud

This section presents the scalability experiments performed on the iCanCloud simulation platform. These experiments focus cloud computing simulation environments by increasing the size of the problem. The aim of those tests exhibits that iCanCloud is able to simulate large cloud computing systems, performing the simulations in a reasonable time frame. Therefore, in order to reach this objective, the amount of memory and the execution time needed for performing the experiments has been measured.

In order to highlight iCanCloud's scalability, two different tests has been performed: the first one, focusing on resources consumed by the experiment due to the environment modelled; the second one, a comparison of iCanCloud with a mature cloud simulator, CloudSim. Results obtained from the experiments have been divided in two different sections, depending on the test performed: the scalability of large size cloud computing architectures, and the comparison with CloudSim.

5.4.1 Scalability of large size cloud computing architectures

We studied the scalability of two different large scale scenarios. These scenarios consists on two data centres. First data centre contains a total of 30720 nodes. The second one has the double size than first scenario, setting up 61440 nodes (see table 5.5).

In order to manage and configure the simulations, the nodes have been grouped by racks. Each rack groups a set of board nodes that, with identical purpose than racks, sets

Environment model configuration	Job model configuration
Racks: 480 and 960	VM per user: 1
Boards per rack: 8	VM cores: 1
Computing nodes per board: 8	VM memory: 1024 MB
Total number of computing nodes: 30720 and 61440	VM disk: 10GB
CPU: Xeon E5 2650L-V3 12 cores	Job Web size: 2.1 Mb
CPU core speed: 113401 MIPS	Job Processing: 10000MIs
Memory: 32 GB DDR3	Job Hits per hour: 9000
Disk: Maxtor 3TB	Job Uptime limit: 24 hours
Network: Ethernet 10 Gbps	# Web servers: 1
Rated output power of PSU: 1000W	
Hypervisor: Xen	

Table 5.5: Environment configuration of cloud model experiments for analysing scalability

and manages a group of nodes, interconnecting them by a switch. In addition, racks are connected to a network of switches. Besides, tenants behaviour model is defined by purchasing virtual machine instances to deploy the web server application model (see section 5.3.2). The experiment simulate the behaviour of the cloud during 24 hours. The configuration of these experiments is described in table 5.5

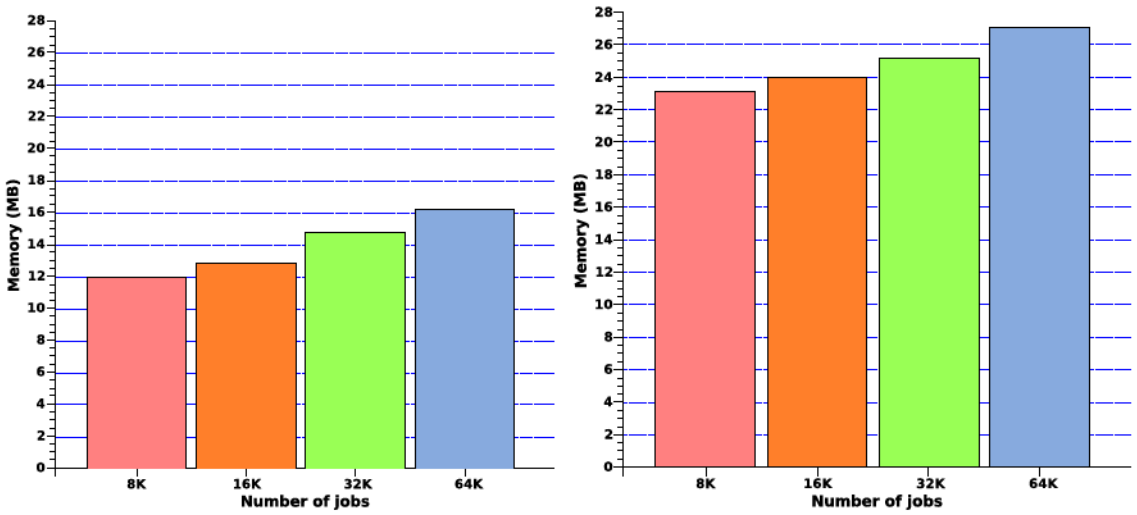
The selection of the number of nodes for the experiments is based on the Top500 [149] list. This list started in 1993, and includes the most powerful supercomputers in the world. It classifies in a simple ranking system the super computers by using parameters, such as teraflops/s, the power consumption, and the number of cores. In order to analyse the capacity to simulate large data centres, the computing nodes of the experiments have been modelled with 12 cores CPUs, reaching the total number cores of the data centre the quantity of 368640 and 737280, and composing on the data centre models with 480 and 960 racks respectively. This simulated number of cores covers all current supercomputers rank from the third.

In order to test the behaviour of the system, different workloads have been modelled. These focuses variations of the number of tenants requesting for resources at the cloud system. Is has been modelled as: 8000, 16000, 32000, and 64000 tenants requesting for a virtual machine to allocate a web server.

These experiments have been executed at several nodes from a cluster, using all cores of each node, which contains two CPU with 8 cores and 64 GB of RAM memory.

Figures 5.12(a) and 5.12(b) show the amount of memory required for the experiments, to simulate a cloud computing environment with a data centre of 30720 computing nodes, and 61440 computing nodes.

First chart 5.12(a) shows that the memory required for the executions increases when the number of jobs increases as well. Each job is defined as a tenant purchasing a VM, and launching a web server application. But a major number of jobs does not have a significant impact on the memory. The same goes with the second experiment, as it is exhibited at second chart 5.12(b). The number of objects deployed by iCanCloud when the workload is increased, does not have as impact as the underlying architecture simulated. The difference between simulate 8000 tenants and 64000 tenants is less than the 25% of the maximum amount of memory required for the experiments.



(a) 30720 Nodes (b) 61440 Nodes

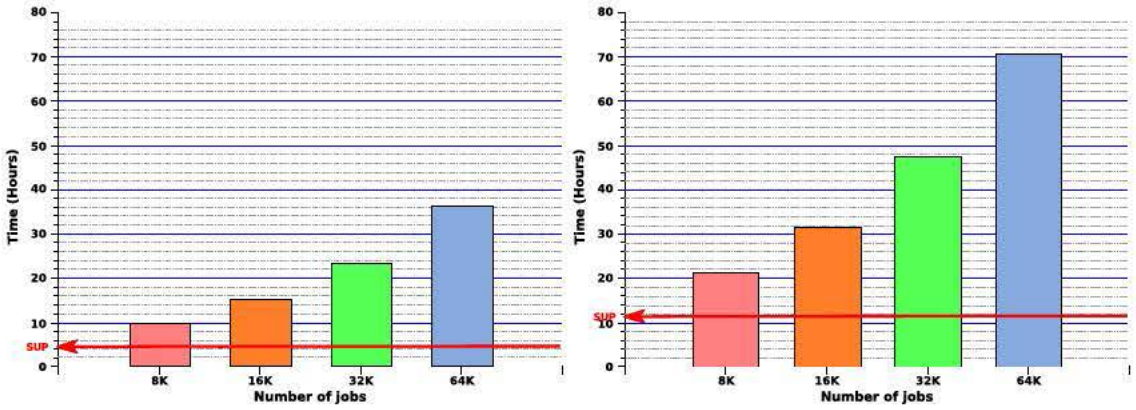
Figure 5.12: Memory usage for simulating large cloud environments experiments

In spite of the memory requirements for deploying simulation models of VMs, tenants, and applications, the simulation of an entire node, due to all the internal components modelled, has the most direct impact on the amount of memory required for executing simulations. This is reflected directly on the comparison between first and second chart, where the variations on the number of jobs do not have as impact, as the increase of the size of the data centre has on the memory system.

The amount of time needed to perform the execution of the scalability experiments, is showed at Figure 5.13(a) and Figure 5.13(b). Each time represented at the charts can be divided as the amount consumed by the set up phase (SUP), and the execution of the experiment. This phase, inherited from OMNeT++, is invoked when the simulation starts. It consists on building the model, inserting the initial events to the set of future events in a data structure (Future Event Set), and creating all the objects defined initially to be allocated in memory. Therefore, in all simulation experiments, all the elements that define the data centre and its management (hardware-software of nodes, network elements, links between elements, and managers), but the tenants, VMs, and jobs, are created at the beginning of the simulation.

Depending on the size of the environment to be simulated, the amount of time required to execute SUP is different. iCanCloud needs 4 hours to set up the environment for a data centre of 30720 nodes, and 11 hours to set up the data centre of 61440 nodes. The set up time is the same for all experiments with identical number of nodes. This is due to the VMs, tenants, and applications models, are created dynamically at runtime. SUP time is represented at the charts as a red arrow.

The total time required for executing the configurations increases when the number of jobs is grows as well. The difference between the amount of time needed for executing 8K and 16K jobs is not as significant, as to execute 32K and 64K configurations, that grows considerably as it is depicted at the chart. This growth is due to the number of active nodes



(a) 30720 Nodes (b) 61440 Nodes

Figure 5.13: Execution time for simulating large cloud environments experiments

that allocates the VMs. When nodes are active, the energy management, hypervisor, and network internals, are active. This entails that many structures are generating events, in order to attend the incoming basic subsystems requests from web servers. This is the main reason for the growth of execution time, as it is exhibited in both charts, Figure 5.13(a) and Figure 5.13(b), where the number of active nodes increases when the number of jobs is major.

The comparison between the time required for simulating the experiments, shows that clearly the time required for simulating 61440 is higher than 30720. In spite of the number of jobs is the same, and also the number of active nodes, the nodes in state off and stand-by state also requires time to be simulated. Those nodes are simulating the energy consumption, and generating events associated to this calculations. These are the main difference between both experiments, producing higher execution times for 61440 nodes than 30470 nodes.

The amount of time required to perform these experiments exhibits the same tendency that the memory usage. However, the time required to execute the simulations grows, when the number of jobs and the size of the data centre is increased.

5.4.2 Comparison with CloudSim

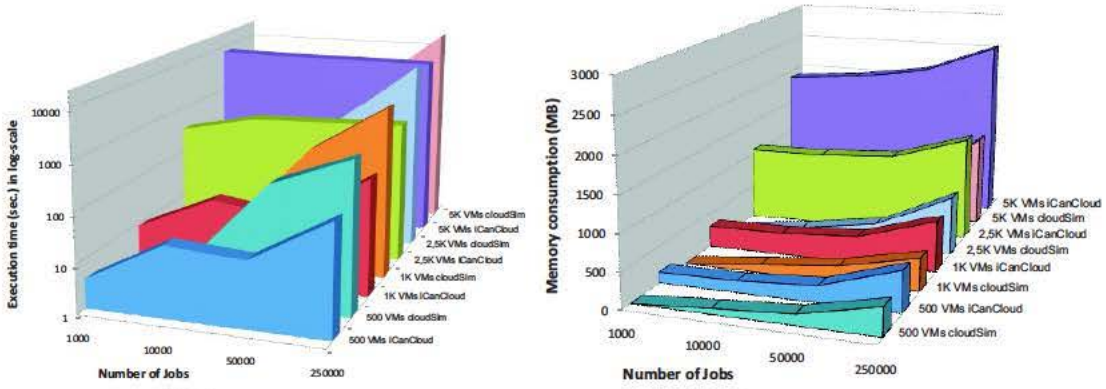
In order to assess the performance of iCanCloud, this experiment has been conducted using it and CloudSim with the purpose to compare both simulation tools. We have chosen CloudSim for this comparison because it is a mature simulation tool which has already been used in a number of research works.

Jobs are modeled in CloudSim by configuring three parameters: (1) input size, (2) processing length, measured in Millions of Instructions (MI), and (3) output size. Input and output sizes refer to the size of input and output files of the job, and is a way to infer the time it takes to execute this job in a cloud resource (whose computing power is measured in Millions of Instructions Per Second – MIPS). Similarly, a new application model has been developed in iCanCloud to execute the same functionality that those jobs

do in CloudSim. Thus, the configuration of jobs used in the experiments described in this section is performed equally in both simulation platforms.

Those experiments use jobs whose input size is 5 MB, output size is 30 MB, and processing length is 1,200,000 MI. Also, cloudlets always utilize all the available CPU capacity. This experiments use VMs having 9,500 MIPS, which simulate a standard small instance type provided by Amazon EC2. The energy models has been disabled for both simulation platforms. Those experiments have been executed using an ASUS computer, model u33J Bamboo, which contains a CPU core i3 and 8 GB of RAM memory.

Figure 5.14 shows the comparison of performance between CloudSim and iCanCloud. Chart 5.14(a) shows the execution time of each experiment, where x-axis shows the number of jobs executed in each experiment, y-axis shows the number of VMs and its type, and z-axis shows the time required to execute each experiment (measured in seconds) using a log-scale.



(a) Execution time

(b) Memory consumption

Figure 5.14: Performance experiments: iCanCloud versus CloudSim

This chart shows that increasing the number of jobs require more execution time in both simulators, which is obvious. Otherwise, increasing the number of VMs has a different impact in each simulator. In iCanCloud, when more than 2500 VMs are used, the number of jobs has not a considerable impact on the execution time. Note that those experiments executed in iCanCloud have a uniform-like shape. That means that almost CPU power consumed by the simulator is processed for managing VMs. Instead, CloudSim's performance depends directly on both parameters (number of VM and number of jobs). In fact, when the value of one of those parameters increases, execution time increases as well. However, besides some experiments where the number of jobs is less or equal to 50000, iCanCloud is faster than CloudSim. Otherwise, in all tests that use 250K jobs iCanCloud is faster. That means that iCanCloud provides better scalability than CloudSim.

Chart 5.14(b) shows the memory consumption of each experiment. In this chart can be appreciated that iCanCloud requires more memory than CloudSim. This is caused because iCanCloud uses a higher level of detail to model each VM instance. As opposed to iCanCloud, which simulates low level details of the cloud system being simulated, CloudSim does not provide in-depth simulation details. Up to 1000 VMs, the amount of memory required by both simulators is similar. But using more than 1000 VMs, the amount of

memory required by iCanCloud grows much faster than CloudSim.

In general, iCanCloud is faster in large scale experiments and provides better scalability, but require more memory than CloudSim.

5.5 Summary

This chapter presents a set of experiments aimed to test the accuracy of the proposed simulation platform. The models for simulating a flexible and scalable simulation platform have been validated, evaluating the simulation platform and the energy model separately.

We present the methodologies which we have followed for validating the framework and the accuracy of the results. The evaluation of the simulator, which has been carried out by conducting a set of experiments on the simulator and comparing them with actual results using instance types provided by Amazon, exhibited that performed tests using iCanCloud obtains practically the same results than real experiments.

Moreover the energy model validation process consisted on performing several experiments using iCanCloud, measuring the energy consumed by individual components of a computing node (CPU, Memory, Network and Disk). Afterwards, the aggregated energy consumption of a complete computing node was calculated by executing the BIPS3D application.

The usefulness of the energy models included into iCanCloud has been demonstrated by modelling two different cloud systems and applications. First, a large data centre that holds VMs that executes web servers have been modelled. Second, a Scientific Cloud environment for executing HPC applications have been simulated. The energy of these systems has been calculated by using different strategies for the unused resources by maintaining these resources in idle and stand-by state.

Finally, scalability experiments are focused on obtaining both the amount of time and memory usage for performing a concrete simulation, depending on the size of the cloud computing environment to be modelled, and the availability of resources to execute the simulations.

Chapter 6

Conclusions and future work

In this thesis we have proposed different strategies for modelling and simulating energy-aware cloud systems. In order to show both the soundness and the applicability of these techniques, different cloud architectures have been simulated for analysing the impact of different workloads on the overall system energy consumption. Hence, the main conclusion obtained from the results obtained from this thesis is that the objectives described in Section 1.3 have been successfully accomplished.

A detailed description of the main contributions of this thesis is presented in this section. Next, some direction of future work, that are currently challenging topics in the scope of this thesis, are described. Finally, the publications resulted from this thesis are shown.

6.1 Main contributions

The main contributions of this thesis can be categorised in three different groups. First, the design and implementation of a simulation platform for modelling and simulating energy aware cloud systems. Second, strategies for simulating and modelling realistic workloads in cloud environments. Finally, optimisation of the underlying architecture in cloud systems in order to balance the trade-offs between performance and energy consumption.

6.1.1 A simulation platform aimed to model and simulate energy aware cloud systems

The corresponding contributions of this group, which cover objectives 1 and 2, are following described:

- A **new simulation platform aimed to model both actual and non-existent cloud systems**. Since currently there exist a lack of tools that provides enough flexibility, in order to model a wide range of posible cloud configurations, and high level of detail, for modelling the energetic consumption of those hardware devices that conforms the underlying architecture of the cloud, we consider that using the proposed simulation platform fits better with the objectives of this thesis than the current simulators focusing on cloud systems.

- The **data center** model that supports the underlying architecture of the simulated clouds can be **fully customised by using a collection of hardware devices**. In order to provide a high level of flexibility, these models can be combined for building a wide range of cloud architectures.

- The **virtualisation layer has been fully modelled and customised** over the data center model to support different cloud scenarios. This layer includes different models of virtual machines and a customisable hypervisor for accordingly managing the requests for each virtualised physical resource, such as CPUs, disks and memories.

- The **energetic consumption of each hardware device** that conforms the architecture of the cloud system has been fully modelled. Furthermore, new models of energetic consumption can be added in order to widen the spectrum of configurations modelled by using the proposed simulation framework.

- The model of a **cloud manager** has been designed in order to **easily include new algorithms for mapping VMs in the available physical machines** that support the cloud. These algorithms may be aimed to saving energy consumption, improving performance and balancing the overall system performance and the required amount of energy for supporting the cloud.

- Finally, a **GUI** has been developed in order to **make easier the process of modelling complete cloud systems**.

6.1.2 Simulating and modelling realistic workloads

The corresponding contributions of this group, which cover objective 3, are following described:

- **Models for executing different workloads, representing a large number of users in cloud systems**. These workloads are defined by the number of tenants that interacts with the cloud and the applications executed by these users. Furthermore, the behaviour of these tenants can be also defined. Basically, this behaviour is modelled by configuring how these tenants arrives to the system and how they request VMs, it being modelled by using different statistical functions.

6.1.3 System optimisation to obtain a good compromise between the overall system performance and energetic consumption

The corresponding contributions of this group which cover the objective 3 are following described:

- A thorough **set of experiments** has been conducted, by using different workloads over different cloud architectures, in order to obtain **the usage costs to perform applications**. Moreover, the **analysis of the impact of energy consumed by physical resources using different system configurations** in a cloud computing environment, has been studied.

6.2 Future work

Although the objectives of this thesis have been successfully fulfilled, currently there are some possible directions that must be explored in order to continue this research. Some of these works are:

- Designing and simulating big data models focusing on cloud based scenarios. Since big data is currently a hot and challenging topic, using the strategies provided in this thesis to represent big data models is a sound and important motivation.
- Including new scheduling algorithms for hypervisors. This allows to analyse and compare the trade-offs between energy consumption and performance of different hardware devices virtualised in the same physical machine.
- Providing new algorithms, focusing on energy saving, for mapping VMs in physical machines. This work is targeted to saving energy in cloud systems without modifying the hardware part of the cloud. Hence, the idea is to optimise the resources used to execute the same number of VMs by using less energy.
- Representing different workloads, based on real traces, gathered from actual cloud systems. The main objective of this work is aimed to reproduce the same behaviour of actual systems, by using different cloud architectures and strategies, in order to obtain a compromise between performance and energy saving.
- Evolve the simulation platform in order to simulate multi cloud architectures. The main aim of this work is to perform new models adding to iCanCloud the capability to simulate multi-cloud environments. The scheduling policies in order to improve clients benefits, ITs and end users, by using different cloud environments is a hot topic that present a new challenge to be solved.

6.3 Publications related with this thesis

This thesis has resulted in the following publications:

- Journals:

  G. G. Castañé, A. Núñez, P. Llopis, and J. Carretero, "E-mc : A formal framework for energy modelling in cloud computing", Simulation Modelling Practice and Theory, vol. 39, pp. 56-75, 2013.

  A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator", Journal of Grid Computing, vol. 10, no. 1, pp. 185-209, 2012.

- Conferences:

  G. G. Castañé, A. Núñez, and J. Carretero, "iCanCloud: A Brief Architecture Overview", 10 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'12) , pp. 853-854, 2012.

G. G. Castañé, A. Núñez, R. Filgueira, and J. Carretero, "Dimensioning Scientific Computing Systems to Improve Performance of Map-Reduce based Applications", Procedia Computer Science, International Conference on Computational Science (ICCS'12), vol. 9, no. 0, pp. 226-235, 2012.

A. Núñez, G. Castañe, J. Vázquez-Poletti, A. Caminero, J. Carretero, and I. Llorente, "Design of a flexible and scalable hypervisor module for simulating cloud computing environments", International Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS'11), pp. 265-270, 2011.

Bibliography

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments, CGE'2008, Workshop on.*, pp. 1–10, Nov 2008.

[2] J. Geelan, "Twenty one experts define Cloud Computing," Jul 2012. Virtualization Electronic Magazine. Available at `http://virtualization.sys-con.com/node/612375`. Accessed on: Mar 29, 2015.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and et al, "Above the Clouds: A Berkeley View of Cloud Computing," *International Journal of High Performance Computing Applications*, vol. 15, pp. 92–101, Feb 2009.

[4] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[5] S. Ried, H. Kisker, P. Matzke, A. Bartels, and M. Lisserman, "Sizing the Cloud," tech. rep., Forrester Research Incorporation, April 2011.

[6] "The Green Grid." http://www.thegreengrid.org/. Accessed on: May 21, 2015.

[7] C. Belady, A. Rawson, and D. Pfleuger, "Green grid data center power efficiency metrics: PUE and DCiE," tech. rep., 2008.

[8] "The Green 500 List." http://www.green500.org. Accessed on: May 22, 2015.

[9] F. Moghaddam, M. Cheriet, and K. Nguyen, "Low Carbon Virtual Private Clouds," in *Cloud Computing, CLOUD'2011. 4th IEEE International Conference on.*, pp. 259–266, IEEE Computer Society, Jul 2011.

[10] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing Cloud Providers' Revenues via Energy Aware Allocation Policies," in *Cloud Computing, CLOUD'2010. 3rd IEEE International Conference on.*, pp. 131–138, Jul 2010.

[11] P. Mahadevan, S. Banerjee, P. Sharma, A. Shah, and P. Ranganathan, "On energy efficiency for enterprise and data center networks," *IEEE Communications Magazine*, vol. 49, pp. 94–100, Ago 2011.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments, GCE'2008. IEEE Workshop on.*, pp. 1–10, Nov 2008.

[13] R. Ge, X. Feng, S. Song, H. Chang, D. Li, and K. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 658–671, May 2010.

[14] "The Economist, A Special Report on Corporate IT," Oct 1st October, 2008. http://www.economist.com/specialReports/showsurvey.cfm?issue=20081025. Accessed on: Jun 02, 2015.

[15] "The Economist, Tanks in the cloud," Dec 2010. http://www.economist.com/node /17797794. Accessed on: Jun 02, 2015.

[16] G. Popek and R. Goldberg, "Formal requirements for virtualizable third generation architectures," *ACM Communications*, vol. 17, pp. 412–421, Jul 1974.

[17] J. Smith and N. Ravi, "The architecture of virtual machines," *Computer Communications*, vol. 38, pp. 32–38, May 2005.

[18] G. Fox, "Keynote Speech: Science Clouds and Their Use in Data Intensive Applications," in *The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications*, ISPA, Jul 2012.

[19] P. Mell and T.Grance, "The NIST Definition of Cloud Computing (v15)," tech. rep., National Institute of Standards and Technology, 2009.

[20] K. Keahey, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications," in *Cloud Computing and Its Applications, CCA2008.*, Oct 2008.

[21] "Here comes HaaS," access on June 2012. http://www.roughtype.com/ archives/2006/03/here_comes_haas.php. Accessed on: Jun 02, 2015.

[22] "PRaaS - Processes as a Service," Dec 2010. http://nauges.typepad.com/ my_weblog/2009/08/praas-process-as-a-service.html. Accessed on: Jun 02, 2015.

[23] D. Machan, "DaaS: The New Information Goldmine," Aug 2009. http://www.economist.com/node/17797794. Accessed on: Jun 04, 2015.

[24] R. Goldberg, "Architecture of virtual machines," in *Virtual computer systems. Workshop on.*, pp. 74–112, ACM, 1973.

[25] "VMWare virtualization software." Web page at `http://www.vmware.com`. . Accessed on: Feb 02, 2015.

[26] "Oracle VM VirtualBox." Web page at `https://www.virtualbox.org/`. Accessed on: May 12, 2015.

[27] "Parallels Workstation." Web page at `hhhp://www.parallels.com/Workstation`. Accessed on: May 14, 2015.

[28] "QEMU - Open source processor emulator." Web page at `www.qemu.org`. Accessed on: May 12, 2015.

[29] "The XEN Hypervisor." Web page at `http://www.xen.org`. Date of last access: 2th February, 2015.

[30] "Kernel based Virtual Machine - KVM." Web page at `http://www.linux-kvm.org`. Accessed on: Feb 27, 2015.

[31] "VMWare ESXi and ESX." Web page at `http://www.vmware.com/products/vsphere/esxi-and-esx/index.html`. Accessed on: May 12, 2015.

[32] "Microsoft Hyper-V Server 2012." Web page at `http://www.microsoft.com/en-us/server-cloud/hyper-v-server/`. Accessed on: May 12, 2015.

[33] C. S. Alliance, "The Notorious Nine," tech. rep., Cloud Security Alliance, Feb 2013.

[34] J. Schad, "Flying Yellow Elephant: Predictable and Efficient MapReduce in the Cloud," in *Very Large Data Bases, VLDB'2010. 36th International Conference on.*, Sep 2010.

[35] L. Barroso, "The Price of Performance," *Queue*, vol. 3, pp. 48–53, Sep 2005.

[36] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computer Survey*, vol. 37, pp. 195–237, Sep 2005.

[37] E. Kursun, S. Ghiasi, and M. Sarrafzadeh, "Transistor Level Budgeting for Power Optimization," in *Quality Electronic Design, ISQED'2004. International Symposium on.*, pp. 116–121, IEEE Computer Society, Mar 2004.

[38] G. Moore, "Transistors to transformations." Web page at `http://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/museum-transistors-to-transformations-brochure.pdf`. Accessed on: Apr 12, 2015.

[39] L. Minas and B. Ellison, *Energy Efficiency for Information Technology: How to Reduce Power Consumption in Servers and Data Centers.* Intel Press, 2009.

[40] B. Davis, "Transistors to transformations." Web page at `http://download.intel.com/pressroom/kits/xeon/5600series/pdf/Xeon\_5600\_PressBriefing.pdf`. Accessed on: Apr 16, 2015.

[41] W. Bircher and L. John, "Analysis of dynamic power management on multi-core processors," in *Supercomputing, ICS'2008. 22th International Conference on.*, pp. 327–338, ACM, Jun 2008.

[42] H. Esmaeilzadeh, T. Cao, Y. Xi, S. Blackburn, and K. McKinley, "Looking back on the language and hardware revolutions: measured power, performance, and scaling," in *Architectural support for programming languages and operating systems, ASPLOS'2011. 16th International conference on.*, pp. 319–332, ACM, March 2011.

[43] X. Fan, W. Weber, and L. Barroso, "Power provisioning for a warehouse-sized computer," in *Computer architecture, ISCA'2007. International Symposium on.*, pp. 13–23, ACM, Jun 2007.

[44] TN-41-01, "Calculating Memory System Power for DDR3," tech. rep., Micron-Technology-Incorporation, 2007.

[45] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved Power Modeling of DDR SDRAMs," in *Digital System Design, DSD'2011. 14th Euromicro Conference on.*, pp. 99–108, IEEE Computer Society, Sep 2011.

[46] JESD79-2F, "DDR2 SDRAM Standard," tech. rep., JEDEC SST Association, 2009.

[47] JESD79-3F, "DDR3 SDRAM Standard," tech. rep., JEDEC SST Association, 2010.

[48] D. Schuhmann, "Strong Showing: High-Performance Power Supply Units." Web page at `http://www.tomshardware.com/reviews/strong-showing,987-38.html`. Accessed on: May 07, 2015.

[49] "80 Plus Certified Power Supplies and Manufacturers." Web page at `http://www.plugloadsolutions.com/80PlusPowerSupplies.aspx`. . Accessed on: Mar 27, 2015.

[50] D. Molaro, H. Payer, and D. Le-Moal, "Tempo: Disk drive power consumption characterization and modeling," in *Consumer Electronics, ISCE'2009. 31th International Symposium on.*, pp. 246–250, May 2009.

[51] A. Hylick, R. Sohan, A. C. Rice, and B. Jones, "An Analysis of Hard Drive Energy Consumption," in *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS'2008. 16th International Symposium on* (E. L. Miller and C. L. Williamson, eds.), pp. 103–112, IEEE Computer Society, Sep 2008.

[52] *A Methodology for generating disk drive energy models using performance data*, Oct 2009.

[53] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power Awareness in Network Design and Routing," in *Computer Communications, INFOCOM'2008. 27th IEEE Conference on.*, pp. 457–465, Apr 2008.

[54] "CISCO systems." Web page at `http://www.cisco.com`. Accessed on: Jun 02, 2015.

[55] "HP Networking." Web page at `http://h17007.www1.hp.com/us/en/`. Accessed on: Apr 29, 2015.

[56] "Brocade - Mission-Critical Networks." Web page at `http://www.brocade.com`. Accessed on: Jun 02, 2015.

[57] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A Power Benchmarking Framework for Network Devices," in *Lecture Notes in Computer Science, NETWORKING 2009.* (L. Fratta, H. Schulzrinne, Y. Takahashi, and O. Spaniol, eds.), vol. 5550, pp. 795–808, Springer Berlin / Heidelberg, Mar 2009.

[58] L. Barroso and U. Holzle, "The Case for Energy-Proportional Computing," *Computer Communications*, vol. 40, pp. 33–37, Dec 2007.

[59] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport," *Proceedings of the IEEE*, vol. 99, pp. 149–167, Jan 2011.

[60] J. Kenneth, C. Gunaratne, B. Nordman, and A. George, "The next frontier for communications networks: power management," *Computer Communications*, vol. 27, pp. 1758–1770, Dec 2004.

[61] Y. Shang, D. Li, and M. Xu, "A Comparison Study of Energy Proportionality of Data Center Network Architectures," in *Distributed Computing Systems Workshops, ICDCSW'2009. 29th International Conference on.*, pp. 1–7, Jun 2012.

[62] D. Abts, M. Marty, P. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Computer Architecture, ISCA'2010. 37th International symposium on.*, pp. 338–347, ACM, Jun 2010.

[63] I. McLean and K. Christensen, "Reducing Energy Use: A Dual-Channel Link," *IEEE Communications Letters*, vol. 16, pp. 411–413, Mar 2012.

[64] Y. Shang, D. Li, and M. Xu, "Energy-aware routing in data center network," in *Green Networking. 1st ACM SIGCOMM Workshop on.*, pp. 1–8, Aug 2010.

[65] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and Myatt, "Best Practices for Data Centers: Results from Benchmarking 22 Data Centers.," in *Energy Efficiency in Buildings, ACEEE Summer Study on.*, Aug 2006.

[66] M. Seymour, C. Aldham, M. Warner, and H. Moezzi, "The Increasing Challenge of Data Center Design and Management: Is CFD a Must?," Dec 2012.

[67] R. Zhou, W. Zhikui, C. Bash, and A. McReynolds, "Data center cooling management and analysis - a model based approach," in *Semiconductor Thermal Measurement and Management Symposium, SEMI-THERM'2012. IEEE.*, pp. 98–103, Mar 2012.

[68] R. Zhou, Z. Wang, C. Bash, A. McReynolds, C. Hoover, R. Shih, N. Kumari, and R. Sharma, "A holistic and optimal approach for data center cooling management," in *American Control Conference, ACC'2011.*, pp. 1346–1351, Jul 2011.

[69] M. Iyengar, M. David, P. Parida, V. Kamath, B. Kochuparambil, D. Graybill, M. Schultz, M. Gaynes, R. Simons, R. Schmidt, and T. Chainer, "Extreme energy efficiency using water cooled servers inside a chiller-less data center," in *IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, ITherm, pp. 137–149, Jun 2012.

[70] R. Miller, "Sea-Cooled Data Center Heats Homes in Helsinki," *Data Center Knowledge*, Sep 2011.

[71] "Barcelona Supercomputing Center." Web page at `http://www.bsc.es/marenostrum-support-services`. Accessed on: Mar 29, 2015.

[72] L. Benini, A. Bogliolo, and G. De-Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 8, pp. 299–316, Jun 2000.

[73] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," *International Conference on Virtual Execution Environments*, vol. 35, pp. 103–116, Oct 2001.

[74] K. Rajamani and C. Lefurgy, "On evaluating request-distribution schemes for saving energy in server clusters," in *Performance Analysis of Systems and Software, ISPASS'2003. IEEE International Symposium on.*, pp. 111–122, Mar 2003.

[75] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos, "Online strategies for high-performance power-aware thread execution on emerging multiprocessors," in *Parallel and Distributed Processing Symposium, IPDPS'2006. IEEE International.*, p. 8 pp., Apr 2006.

[76] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting redundancy to conserve energy in storage systems," *SIGMETRICS Performance Evaluation Review*, vol. 34, pp. 15–26, Jun 2006.

[77] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning, "PARAID: A gear-shifting power-aware RAID," *Transactions on Storage*, vol. 3, Oct 2007.

[78] S. Yin, Y. Tian, J. Xie, X. Qin, M. Alghamdi, X. Ruan, and M. Qiu, "Reliability analysis of an energy-aware RAID system," in *Performance Computing and Communications Conference, IPCCC'2011. 30th IEEE International.*, pp. 1–8, Nov 2011.

[79] J. Wang, H. Zhu, and D. Li, "eRAID: Conserving Energy in Conventional Disk-Based RAID System," *IEEE Transactions on Computers*, vol. 57, pp. 359–374, Mar 2008.

[80] X. Liao, S. Bai, Y. Wang, and S. Hu, "ISRA-Based Grouping: A Disk Reorganization Approach for Disk Energy Conservation and Disk Performance Enhancement," *IEEE Transactions on Computers*, vol. 60, pp. 292–304, Feb 2011.

[81] T. Bostoen, S. Mullender, and Y. Berbers, "Power-reduction techniques for datacenter storage systems," *ACM Computer Survey*, vol. 45, pp. 33:1–33:38, Jun 2013.

[82] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, Apr 1992.

[83] C. Ravishankar, S. Ananthanarayanan, S. Garg, and A. Kennings, "Analysis and evaluation of greedy thread swapping based dynamic power management for MPSoC platforms," in *Quality Electronic Design, ISQED'2012. 16th International Symposium on .*, pp. 617–624, Mar 2012.

[84] J. Khan, S. Bilavarn, and C. Belleudy, "Energy analysis of a DVFS based power strategy on ARM platforms," in *Faible Tension Faible Consommation, FTFC'2012.*, pp. 1–4, Jun 2012.

[85] Y. Lu, C. Lai, and Y. Huang, "Parallelization of DVFS-enabled H.264/AVC Decoder on Heterogeneous Multi-core Platform," in *Computer, Consumer and Control, IS3C'2012. International Symposium on.*, pp. 157–160, Jun 2012.

[86] S. Hoppner, C. Shao, H. Eisenreich, G. Ellguth, M. Ander, and R. Schuffny, "A power management architecture for fast per-core DVFS in heterogeneous MPSoCs," in *Circuits and Systems, ISCAS'2012. IEEE International Symposium on.*, pp. 261–264, May 2012.

[87] Y. Guihai, Y. Li, Y. Han, X. Li, M. Guo, and X. Liang, "AgileRegulator: A hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture," in *High Performance Computer Architecture, HPCA'2012. 18th IEEE International Symposium on.*, pp. 1–12, Feb 2012.

[88] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," in *Quality Electronic Design, ISQED'2012. 13th International Symposium on.*, pp. 747–754, Mar 2012.

[89] D. Kim, S. Kim, and S. Kim, "Dynamic power management for Cluster system," in *Advanced Communication Technology, ICACT'2012. 14th International Conference on.*, pp. 107–110, Feb 2012.

[90] "iASL: ACPI Source Language Optimizing Compuler and Dissasembler. Revision 5.02.Jun.20." Web page at `http://www.acpica.org/download/aslcompuler.pdf`. Accessed on: 3rd January, 2015.

[91] M. Powell and B. Miller, "Process migration in DEMOS/MP," in *Operating systems principles, SOSP'1983. 9th Symposium on*, pp. 110–119, ACM, Oct 1983.

[92] E. Jul, H. Levy, N. Hutchinson, and A. Black, "Fine-grained mobility in the Emerald system," *ACM Transactions on Computer Systems*, vol. 6, pp. 109–133, Feb 1988.

[93] M. Theimer, M. Marvin, K. Lantz, and D. Cheriton, "Preemptable remote execution facilities for the V-system," *International Conference on Virtual Execution Environments*, vol. 19, pp. 2–12, Dec 1985.

[94] A. Spellmann, K. Erickson, and J. Reynolds, "Server consolidation using performance modeling," *IT Professional*, vol. 5, pp. 31–36, Sep 2003.

[95] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Power aware computing and systems, HotPower'2008. 8th Conference on.*, pp. 10–10, USENIX Association, Dec 2008.

[96] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Networked Systems Design and Implementation, NSDI'2005. 6th Conference on Symposium on.*, pp. 273–286, USENIX Association, Ago 2005.

[97] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Genererations Computer Systems*, vol. 22, pp. 901–907, Oct 2006.

[98] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Virtual execution environments, VEE'2007. International conference on.*, pp. 169–179, ACM, Jun 2007.

[99] H. Liu, C. Xu, C. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *High performance distributed computing, HPDC'2011. 20th International symposium on.*, pp. 171–182, ACM, Jun 2011.

[100] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power Consumption of Virtual Machine Live Migration in Clouds," in *Communications and Mobile Computing, CMC'2011. 3rd International Conference on.*, pp. 122–125, IEEE Computer Society, Apr 2011.

[101] W. Liu and T. Fan, "Live migration of virtual machine based on recovering system and CPU scheduling," in *Information Technology and Artificial Intelligence Conference, ITAIC'2011. 6th IEEE Joint International.*, pp. 303–307, Aug. 2011.

[102] J. Yang, "Key Technologies and Optimization for Dynamic Migration of Virtual Machines in Cloud Computing," in *Intelligent System Design and Engineering Application, ISDEA'2012. 2nd International Conference on.*, pp. 643–647, Jan 2012.

[103] H. Liu, H. Jin, X. Liao, C. Yu, and C. Xu, "Live Virtual Machine Migration via Asynchronous Replication and State Synchronization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1986–1999, Dec 2011.

[104] M. Chen, H. Zhang, Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," in *Integrated Network Management, IFIP'2011. 29th International Symposium on*, pp. 594–601, May 2011.

[105] S. Bose, S. Brock, R. Skeoch, and S. Rao, "CloudSpider: Combining Replication with Scheduling for Optimizing Live Migration of Virtual Machines across Wide Area Networks," in *Cluster, Cloud and Grid Computing,CCGrid'2011. 11th IEEE/ACM International Symposium on.*, pp. 13–22, May 2011.

[106] R. Shannon, "Introduction to the art and science of simulation," in *The Winter Simulation Conference, WSC'1998.*, pp. 7–14, IEEE Computer Society Press, Dec 1998.

[107] E. Winsberg, *Science in the Age of Computer Simulation.* University of Chicago Press, 2010.

[108] "The Network Simulator, NS-2." Web page at `http://www.isi.edu/nsnam/ns/`. Accessed on: Feb 27, 2015.

[109] J. Liu and D. Nicol, *DaSSF 3.1 User's Manual.* Dartmouth College, Apr 2001.

[110] A. Varga, "The OMNeT++ discrete event simulation system," in *European Simulation Multiconference, ESM'2001*, Jun 2001.

[111] Modeler, OPNET, "OPNET Technologies Inc." Web page at `http://www.opnet.com/`. Accesed on: 18th May 2015, 2009.

[112] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer Communications*, vol. 35, pp. 50–58, Feb 2002.

[113] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, pp. 92–99, Nov 2005.

[114] N. Hardavellas, S. Somogyi, T. Wenisch, E. Wunderlich, S. Chen, J. .Kim, B. Falsafi, J. Hoe, and A. Nowatzyk, "Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *SIGMETRICS Performance Evaluation Review*, vol. 31, pp. 31–35, Mar 2004.

[115] A. Núñez, J. Fernández, J. García, F. García, and J. Carretero, "New techniques for simulating high performance MPI applications on large storage networks," *Journal of Supercomputing*, vol. 51, pp. 40–57, Sep 2010.

[116] C. Kesselman and I. Foster, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Nov 1998.

[117] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency & Computation: Practice & Experience*, vol. 14, pp. 1175–1220, Dec 2002.

[118] W. Bell, G. David, C. Capozza, L. Capozza, A. Millar, K. Stockinger, and F. Zini, "Simulation of Dynamic Grid Replication Strategies in OptorSim," in *Grid Computing, Grid'2002. 3rd International Workshop on.*, Nov 2002.

[119] X. Liu, *Scalable Online Simulation for Modeling Grid Dynamics*. PhD thesis, Univ. of California at San Diego, 2004.

[120] C. Dumitrescu and I. Foster, "GangSim:a simulator for grid scheduling studies," in *Cluster Computing and Grid, CCGrid'2005. 5th IEEE/ACM International Symposium on.*, vol. 2, May 2005.

[121] R. Buyya, R. Ranjan, and R. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities," in *High Performance Computing and Simulation Conference, HPCS'2009.*, Jun 2009.

[122] S. Lim, B. Sharma, G. Nam, E. Kim, and C. Das, "MDCSim: A multi-tier data center simulation platform," in *Cluster Computing and Workshops, CLUSTER'2009. International Conference on.*, Sep 2009.

[123] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan, "GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers," in *Global Telecommunications Conference, GLOBECOM'2010.*, pp. 1–5, Dec 2010.

[124] H. Casanova, "Simgrid: a Toolkit for the Simulation of Application Scheduling," in *Cluster, Cloud and Grid Computing,CCGrid'2001. 1st IEEE/ACM International Symposium on.*, pp. 430–437, May 2001.

[125] J. Wu, X. Zhao, X. Sui, and X. Yang, "PVMsim: A parallel simulation platform to evaluate virtual machines," in *Future Computer and Communication, ICFCC'2010. International Conference on.*, vol. 2, pp. V2–551–V2–556, May 2010.

[126] A. García-Guirado, R. Fernández-Pascual, and J. M. García, "Virtual-GEMS: An Infrastructure To Simulate Virtual Machines," in *Modelling, benchmarking and simulation, MoBS'2009. 5th workshop on.*, Jun 2009.

[127] I. Sriram and D. Cliff, "SPECI-2 - An Open-source Framework for Predictive Simulation of Cloud-scale Data-centres," in *Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH'2011. 2nd International Conference on.*, pp. 418–421, Jul 2011.

[128] K. Kim, A. Beloglazov, and R. Buyya, "Power-aware Provisioning of Cloud Resources for Real-time Services," in *Middleware for Grids, Clouds and e-Science, MGC'2009. 7th International Workshop on.*, Dec 2009.

[129] R. Calheiros, R. Buyya, and C. D. Rose, "Building an automated and self-configurable emulation testbed for grid applications," *Software: Practice and Experience*, vol. 40, pp. 405–429, Apr 2010.

[130] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges," in *Parallel and Distributed Processing Techniques and Applications, PDPTA'2010. 16th International Conference on.*, Jul 2010.

[131] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating Data Grids: An extension to GridSim," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 1591–1609, Sep 2008.

[132] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *Advanced Information Networking and Applications, AINA'2010. 24th International Conference on.*, Mar 2010.

[133] "CSIM Development Toolkit for Simulation and Modeling." Web page at `http://www.mesquite.com/`. Date of last access:27th July, 2014.

[134] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," in *Computer Architecture and High Performance Computing, SBAC-PAD'2007. 19th International Symposium on.*, pp. 62–68, Oct 2007.

[135] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. D. Hill, and D. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Computing Architecture News*, vol. 33, pp. 92–95, Nov 2005.

[136] I. Sriram, "SPECI, a Simulation Tool Exploring Cloud-Scale Data Centres.," in *CloudCom* (M. G. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of *Lecture Notes in Computer Science*, pp. 381–392, Springer, 2009.

[137] A. H. Buss, "Simkit: component based simulation modeling with Simkit.," in *Winter Simulation Conference, WSC'2002.* (J. L. Snowdon and J. M. Charnes, eds.), pp. 243–249, ACM, Dec 2002.

[138] A. Núñez, J. Fernández, R. Filgueira, F. García, and J. Carretero, "SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications," *Simulation Modelling Practice and Theory*, vol. 20, pp. 12–32, Jan 2012.

[139] A. Nuñez, J. Vázquez-Poletti, A. Caminero, G. Castañé, J. Carretero, and I. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator," *Journal of Grid Computing*, vol. 10, pp. 185–209, Mar 2012.

[140] A. Núñez, J. Fernández, J. D. Garcia, L. Prada, and J. Carretero, "New Techniques for Modeling File Data Distribution on Storage Nodes," in *44th Annual Simulation symposium, ANSS'2008*, (Washington, DC, USA), pp. 175–182, IEEE Computer Society, Mar 2008.

[141] A. Varga, "The INET Framework," 2007. http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/INETFramework. Accessed on: Mar 12, 2015.

[142] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. Maestro, "IEEE 802.3az: the road to energy efficient ethernet," *Communications Magazine, IEEE*, vol. 48, pp. 50–56, Nov 2010.

[143] S. Schlesinger, "Terminology for model credibility," *Transactions of the society for modeling and simulation international*, vol. 32, pp. 103–104, Mar 1979.

[144] J. Vázquez-Poletti, G. Barderas, I. Llorente, and P. Romero, "A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure.," vol. 7133, pp. 33–42, 2010.

[145] A. Harri, V. Linkin, K. Pichkadze, W. Schmidt, R. Pellinen, A. Lipatov, L. Vazquez, H. Guerrero, M. Uspensky, and J. Polkko, "MMPM-Mars MetNet Pre-cursor Mission," in *European Geosciences Union General Assembly*, Apr 2008.

[146] C. Zehan, Z. Yan, B. Yungang, and C. Mingyu, "A fine-grained component-level power measurement method," in *Green Computing Conference and Workshops, IGCC'2011. International*, pp. 1–6, IEEE, Jul 2011.

[147] H. Chen, M. Song, J. Song, A. Gavrilovska, K. Schwan, and M. Kesavan, "CACM: Current-aware capacity management in consolidated server enclosures," in *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–6, Jul 2011.

[148] K. Gyllstrom and M. Moens, "Surfin' Wikipedia: an analysis of the Wikipedia (non-random) surfer's behavior from aggregate access data," in *Context Symposium, IIilx'2012. 4th Information Interaction in* , pp. 155–163, ACM, Aug 2012.

[149] "Top 500 supercomputer list." http://www.top500.org/. Accessed on: May 31, 2015.